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

**Jia Liang**[2] **Liangming Pan**[1*]
[1]MOE Key Laboratory of Computational Linguistics,
School of Computer Science, Peking University
[2]Institute for Computational and Mathematical Engineering, Stanford University
jialiang2015@gmail.com liangmingpan@pku.edu.cn
Project: https://github.com/jialiang19/latent-cot-thinking

## Abstract

Latent Chain-of-Thought aims to enable step-by-step computation without emitting long rationales, yet its internal mechanisms remain unclear. We study CODI, a continuous-thought teacher–student distillation model, on strictly sequential polynomial-iteration tasks with known intermediate states. Using logit-lens decoding, linear probes, attention analysis, and activation patching, we localize intermediate-state representations and trace how they are routed to the final readout. In short-horizon, low-hop tasks, CODI forms faithful bridge states across latent-thought positions, while the final input follows a separate near-direct route; predictions arise through late fusion at the answer readout. As task depth and difficulty increase, however, CODI does not reliably sustain a full latent rollout: it either compresses computation into a partial late-intermediate pathway or, in harder regimes, loses the latent reasoning signature altogether. To explain this transition, we show theoretically that the task's algebraic structure controls its effective memory: compressible regimes support late-bottleneck reasoning, while incompressible regimes preserve full-history dependence and destabilize latent rollouts. We further generalize this framework to finite-state function composition, where transition rank controls compressibility, and empirically validate the same qualitative patterns beyond modular arithmetic. Extending our analysis to Coconut, a latent-CoT model trained under a different paradigm, reveals similar patterns of partial late-intermediate decodability and reasoning collapse across latent-thought positions. Overall, our findings demonstrate how task compressibility can alter the underlying latent-reasoning mechanisms.

## 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 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",

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

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. 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), leaving the mechanisms and step-by-step faithfulness of latent chain-of-thought comparatively underexplored. 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.

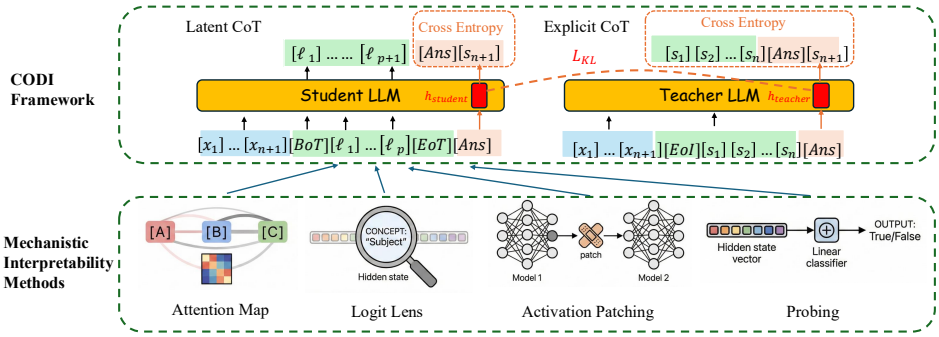



Figure 1: **Mechanistic study of CODI on sequential reasoning tasks.** *Top:* the CODI training setup used for our polynomial iteration task. *Bottom:* the four mechanistic interpretability methods we use to analyze the student model's internal computations.

In this paper, we give a mechanistic account of how latent CoT solves sequential multi-hop algorithmic tasks when computation is routed through a latent "thought" channel. Using logit-lens (nostalgebraist, 2020) decoding, linear probes (Alain and 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: In the default composite-modulus polynomial setting, on short-horizon, low hop tasks (2–3 hops), 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 in this setting ($n \geq 4$), CODI rarely realizes a full latent rollout; instead, the latent channel shifts toward a partial-reasoning, late-bottleneck trace that retains only the final one or two intermediates.

Our polynomial-iteration tasks operate over integers modulo $m$. A striking empirical pattern emerges: mechanistic signatures of latent step-by-step reasoning persist across many composite moduli but largely vanish for prime moduli, where accuracy drops and intermediate-state decodability weakens or disappears. This discontinuity suggests that latent-CoT behavior depends not only on task length, but also on effective compressibility: composite moduli introduce many-to-one updates that can erase earlier information and make late-bottleneck strategies viable, whereas prime moduli make nonzero updates bijective, preserving dependence on the full input history and stressing the fixed

latent budget. **Our contributions are threefold**: First, we causally trace CODI-style latent CoT on strictly sequential tasks, identifying where intermediate states emerge and how they are routed to the final answer. Second, we formalize the above intuition through a theory of task-level compressibility, showing how many-to-one contractions reduce the amount of prefix information needed to predict the final state, whereas bijective dynamics preserve long-range dependence and stress a fixed latent budget. Third, we validate this account beyond the CODI-specific training framework using the Coconut architecture, and beyond modular arithmetic using finite-state function composition (FSC). In FSC, transition rank directly controls compressibility, and varying the relative proportions of compressive and bijective transitions reproduces the same mechanistic transition.

## 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., 2025c). 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 and 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 and 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 reasoning-related representations can be steered or decoded (Zhang and Viteri, 2024; Wang et al., 2024b), these results are primarily correlational and do not directly reveal the underlying mechanism. More recent work highlights the theoretical advantages of latent Chain-of-Thought (CoT) for parallel exploration and graph-based search (Zhu et al., 2025a; Gozeten et al., 2025; Xu and Sato, 2025) and uncovers probing inconsistencies in looped transformers (Lu et al., 2025). While other recent mechanistic studies of latent-CoT models have identified structured scratchpad–computation cycles (Goyal et al., 2025) and shortcut behavior (Zhang et al., 2025b), recent and concurrent studies of Coconut and CODI have further investigated the heterogeneous causal roles of latent steps and highlighted that intermediate-state decodability alone does not establish faithful computation (Li et al., 2026; Aswal et al., 2026; Qiu et al., 2026). However, how task structure shapes the emergence and breakdown of latent reasoning on sequential tasks remains less understood. We complement these studies by providing a mechanistic analysis of CODI on controlled sequential tasks and characterizing how task depth, difficulty, and compressibility shape the transition from faithful step-by-step computation to partial late-bottleneck reasoning or the breakdown of decodable latent trajectories. We provide additional discussion of recent and concurrent work on latent-reasoning mechanisms and supervision in Appendix P.

## 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

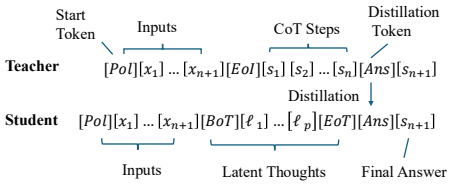



Figure 2: **Polynomial tasks for training CODI.** The teacher generates an explicit CoT trace, while the student answers after a fixed-length latent-thought trajectory.

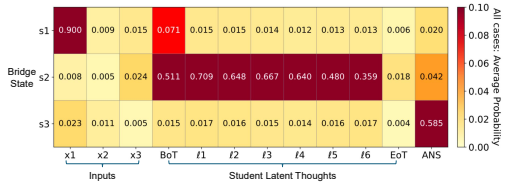



Figure 3: **Logit lens on intermediate states** $s_1, s_2, s_3$**.** The bridge state $s_2$ becomes decodable during latent computation. Each cell show average decoding probability across all layers and all test inputs.

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 $\ell_1$ 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 $x_1, \ldots, x_{n+1} \in \mathbb{Z}_m$ and generate states $s_1, \ldots, s_{n+1}$ by $s_1 = x_1$ and

$$s_t \;=\; s_{t-1}x_t + b \pmod{m}, \qquad t = 2, \ldots, n+1, \tag{1}$$

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).

**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

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

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 *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?

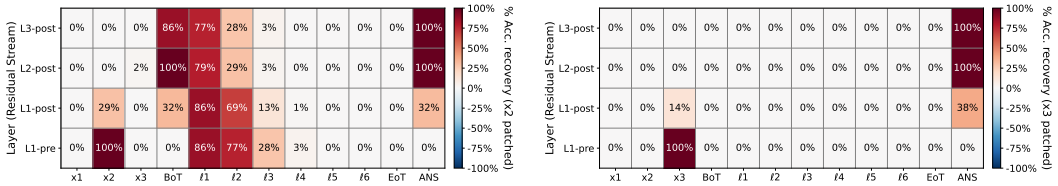
**Attention Analysis.** From the attention maps in Figure 7 (Appendix B), we observe several heads in which the `[Ans]` tokens place high attention weight directly on the $x_3$ token. This pattern is consistent with *a copy-like pathway that routes information from $x_3$ into the residual streams of `[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 is strongly decodable at `[Ans]`, with probe probability near 0.85. **Targeted attention knockouts** show that the bridge-state signal $s_2$ reaches `[Ans]` primarily through the early-to-middle latent positions: simultaneously blocking attention from `[Ans]` to `[BoT]`, all latent positions $\ell_1$–$\ell_6$, and `[EoT]` collapses accuracy to 3.8%. **In leave-one-in ablations**, we block all of these sources except one and measure whether that single source can recover performance. Only $\ell_1$–$\ell_4$ recover most of the accuracy; `[BoT]` and `[EoT]` alone do not. See Appendix G for details.

Together with the logit-lens, probing, the attention analysis suggests a clear division of labor. The final input $x_3$ is routed to `[Ans]` through a largely direct, copy-like pathway, whereas the bridge state $s_2$ is formed and maintained in the latent channel and then read out through the early-to-middle latent positions. The final prediction is therefore produced by late fusion at `[Ans]`, where the residual stream combines the latent bridge-state signal with the directly routed $x_3$ signal before 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 `[BoT]` position and remains large for early latent steps $\ell_1$ and $\ell_2$. In particular, patching at `L2-Post` on the `[BoT]` token restores accuracy to $100\%$; recovery is still around $80\%$ for $\ell_1$ across layers and around $70\%$ for $\ell_2$ 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. **State-swap interventions** provide complementary causal evidence: replacing activations with those from a suffix-matched prompt that shares $x_3$ but differs in $s_2$ flips the prediction primarily at `[BoT]` and early latent steps ($l_1$–$l_2$), and the effect rapidly disappears in later latent positions; see Appendix H for details.

Together, these causal effects indicate that the intermediate state is constructed and used at `[BoT]` 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]`.




(a) Patching for $x_2$-corrupted runs

(b) Patching for $x_3$-corrupted runs

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.

### 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.

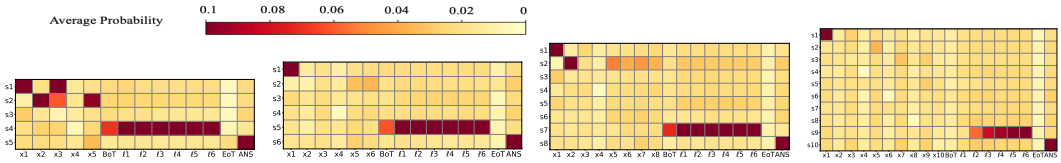
**$n$-hop Polynomial Task, $n \geq 4$.** In the $n$-hop task we have $n+1$ inputs (i.e., $x_1, \ldots, x_{n+1}$) and $n-1$ intermediate bridge states, $s_2, \ldots, s_n$. For $n \geq 4$, increasing hop depth preserves the same two-stream routing pattern but induces a clear *collapse* in intermediate-state 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 8 in Appendix B). However, logit-lens and probing analyses over both input-token positions $x_1, \ldots, x_{n+1}$ and latent positions `[`$\ell_1$`]`–`[`$\ell_6$`]` show that the latent stream does *not* expose a step-by-step chain of intermediates. Among $s_2, \ldots, s_n$, only the late intermediate $s_n$ (sometimes the last two intermediates $s_{n-1}, s_n$ as seen in Appendix I) is reliably decodable, and this decodable signal appears primarily in the latent positions, as shown 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. In general, 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 $(s_1, \ldots, s_n)$. 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]`. Even at a hop count as large as $n = 31$, we continue to observe the previously described partial latent pathway.

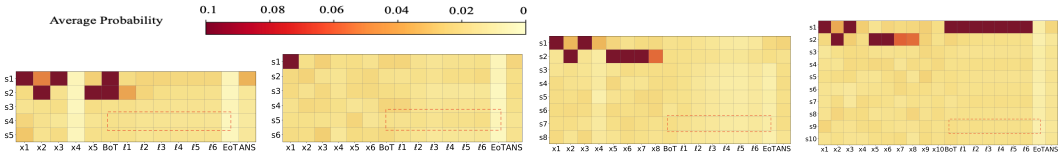
### 4.4 How does task definition affect the observed mechanism?

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 ($m \in \{41, \ldots, 50\}$). Changing $b$ does not qualitatively affect the behaviors reported




(a) 4-hop task (b) 5-hop task (c) 7-hop task (d) 9-hop task

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.




(a) 4-hop task (b) 5-hop task (c) 7-hop task (d) 9-hop task

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

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 in Appendix L) 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.

## 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 $x_1, \ldots, x_T \in \{1, \ldots, m-1\}$, define a sequential state by

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

for $t = 2, \ldots, T$, 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 $(x_1, \ldots, x_{T-k})$.

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

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

*Proof.* See Appendix J.1.

**Composite moduli create many-to-one "contractions."** When $\gcd(x, m) > 1$, multiplication by $x$ collapses information.

**Lemma 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 J.2.

Lemma 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 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

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

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 $x_t \sim \mathrm{Unif}\{1, \dots, m-1\}$. Define the unit probability

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

where $\varphi$ 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 2). More generally, let

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

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

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

Thus, for composite $m$ with $u(m) \ll 1$, there is typically a *recent* contraction event $\tau$ 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 $\tau$).

**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 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 in Appendix L and Figure 6. For composite $m$, non-unit multipliers occur with probability $q(m) > 0$ and induce explicit many-to-one contractions (Lemma 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 Beyond Modular Arithmetic: Finite-State Function Composition

Section 5 suggests that the key factor behind the prime–composite split is not modular arithmetic itself, but whether state updates preserve or compress information. We therefore introduce FSC, a finite-state function-composition task that controls transition-level compressibility.

Let $\mathcal{S} = \{0, \ldots, N-1\}$ be a finite state space and let $\mathcal{F} = \{f_1, \ldots, f_K\}$ be a fixed library of functions $f_i : \mathcal{S} \to \mathcal{S}$, sampled once and reused across train and test examples. Given an initial state $s_0$ and operation indices $\sigma_1, \ldots, \sigma_T$, the trajectory is

$$s_t = f_{\sigma_t}(s_{t-1}), \qquad t = 1, \ldots, T.$$

The model receives $(s_0, \sigma_1, \ldots, \sigma_T)$ and predicts the intermediate trajectory and final state. FSC controls compressibility through the transition rank $r_i := |\operatorname{Im}(f_i)|$. A fraction $\alpha$ of the library is sampled as bijections, with $r_i = N$, while the remaining functions are compressive maps with $r_i = m_i < N$. Thus FSC removes primes, composites, and gcd structure, while retaining the central variable from Section 5: whether an update preserves or contracts state information. See Appendix M.1 for the complete dataset definition and construction details.

The FSC theory generalizes the polynomial prime–composite analysis by replacing algebraic gcd structure with transition rank. Fully bijective libraries preserve all state distinctions under composition, making long-horizon prediction require sustained state propagation. Compressive transitions, by contrast, reduce the number of distinguishable prefix states, and later deterministic updates cannot recover the lost information. Under uniform operation sampling, the terminal all-bijective suffix has geometric tail, so compressive regimes typically contain recent rank contractions that make late-bottleneck reasoning viable. See Appendix M.2 for full statements and proofs.

FSC reproduces the main polynomial-task phenomena. On short-horizon tasks, CODI forms causally relevant bridge states in the latent trajectory. On longer compressive FSC tasks, it shifts to a late-bottleneck strategy in which only the final bridge state remains decodable, mirroring composite-modulus polynomial tasks. In the fully bijective regime ($\alpha = 1$), this late-intermediate trace disappears on long horizons, paralleling the prime-modulus failure mode. Full experimental details are provided in Appendix M.3. A combined analysis of the polynomial and FSC tasks suggests that the transition from faithful latent rollouts to partial or collapsed reasoning is not determined by hop count alone, but by the effective task difficulty relative to the model and training budget, with compressibility shifting this transition point. See Appendix N for details.

Moreover, FSC captures deterministic finite-state, operation-indexed tasks: any update rule $z_t = \delta(z_{t-1}, a_t)$ can be represented by defining one transition function $f_i(z) = \delta(z, a_i)$ per operation. This includes finite automata, lookup-table systems, bounded finite-register systems, and the modular polynomial task itself. FSC does not directly cover unbounded-memory, continuous-state, stochastic, or richly structured-operation settings without additional modeling choices. Thus, FSC is a controlled abstraction for testing whether the compressibility mechanism observed in polynomial iteration persists beyond modular arithmetic; details are in Appendix M.4.

## 7 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 K.

## 8 Extension to Coconut

To investigate whether the observed latent-reasoning mechanisms are specific to CODI's self-distillation framework, we extend our analysis to Coconut (Hao et al., 2024), a latent-CoT model trained through progressive internalization of explicit reasoning steps. While both CODI and Coconut generate continuous thoughts through autoregressive hidden-state propagation, they differ in their training strategies: CODI employs joint teacher–student training with feature-space self-distillation, whereas Coconut progressively replaces explicit CoT steps with continuous thoughts.

**Experimental Setup.** We train a 3-layer GPT-2 model ($d_{\text{model}} = 128$, 2 attention heads) on the same polynomial-iteration task described in Section 3, using Coconut's curriculum-learning strategy to

progressively replace explicit CoT steps with continuous latent thoughts. We evaluate polynomial tasks with $n \in \{5, 6, 7\}$ and moduli $m \in \{41, 50\}$. Full details are provided in Appendix O.

**Composite Modulus ($m = 50$).** Across $n \in \{5, 6, 7\}$, Coconut does not exhibit a complete, faithfully decodable step-by-step latent trajectory. Logit-lens analysis in Figure 21 shows that several intermediate states, such as $s_3$ and $s_4$ for $n = 5$, $s_4$ for $n = 6$, and $s_3$, $s_4$, and $s_6$ for $n = 7$, remain difficult to decode from the latent thoughts. Instead, the latent trajectory, $l_1, ..., l_n$, primarily decodes the final intermediate state $s_n$. This pattern is qualitatively similar to the late-bottleneck behavior observed in CODI (Figure 5). Although Coconut appears to make more intermediate states decodable than CODI in some settings, neither model consistently exposes the complete sequence of ground-truth intermediate states within its latent thoughts on these longer tasks. These findings suggest that the incomplete latent reasoning trajectories observed in CODI are not unique to its self-distillation training framework.

These findings are consistent with the theoretical framework developed in Section 5: under composite moduli, many-to-one contractions (Lemma 2) induce a compressible task structure that favors partial, late-intermediate reasoning. This provides a plausible explanation for why Coconut's latent thought positions, $l_1, ..., l_{n-1}$, primarily expose the final intermediate state $s_n$, while earlier intermediate states are not consistently decodable.

**Prime Modulus ($m = 41$).** In contrast, for prime-modulus tasks across $n \in \{5, 6, 7\}$, the latent reasoning signature largely disappears. Logit-lens analysis in Figure 22 reveals that even the final intermediate state $s_n$ is difficult to decode from the latent thoughts, indicating a breakdown of the late-bottleneck pattern. This behavior closely resembles the reasoning collapse observed in CODI under prime moduli (Figure 6). This observation is also consistent with our theoretical analysis. Under prime moduli with nonzero multipliers, all state transitions are bijective and therefore cannot erase earlier state distinctions through many-to-one contractions. Consequently, the computation lacks the contraction-based shortcuts available under composite moduli, making both sustained step-by-step reasoning and late-bottleneck compression more challenging under limited model capacity and latent computation.

**Takeaway.** Despite different training paradigms, Coconut and CODI exhibit similar patterns of partial late-bottleneck-like reasoning under composite moduli and collapsed reasoning under prime moduli, suggesting that these behaviors extend beyond CODI's training framework.

## 9 Conclusion

We mechanistically analyzed CODI on polynomial-iteration tasks. Under composite moduli, CODI exhibits faithful step-by-step reasoning on short-horizon tasks but shifts to partial, late-intermediate rollouts on longer horizons. Under prime moduli, however, its performance degrades sharply and the latent reasoning signature largely disappears. Our theory explains this distinction through task *compressibility*: composite-modulus dynamics admit many-to-one contractions that enable late-bottleneck solutions, whereas prime-modulus dynamics preserve state distinctions and require sustained information propagation. We further validate this framework on finite-state function-composition tasks, where compressive and bijective transitions yield partial and collapsed reasoning signatures, respectively. Coconut exhibits similar patterns under composite and prime moduli despite its different training paradigm. Together, these findings highlight the importance of evaluating latent-CoT models along both task horizon and compressibility. Additional discussion of latent CoT comparisons with non-CoT and CoT models, computational resources, and broader impacts is provided in Appendix L, Appendix Q, and Appendix R, respectively.

**Limitations & future directions.** While our study primarily focuses on CODI, with additional experiments on Coconut, further work is needed to determine how broadly the observed mechanisms extend to other latent-reasoning architectures, such as looped transformers, recurrent latent-state models, or hybrid latent-text approaches. In addition, our polynomial and finite-state sequential tasks provide ground-truth intermediate states and allow us to vary compressibility precisely, but they remain synthetic abstractions of reasoning. An important direction for future work is to test whether the *compressibility principle* predicts latent-reasoning behavior in more naturalistic datasets, where ambiguity, redundancy, and many-to-one structure arise from language or real-world tasks rather than from a hand-designed construction.

## Acknowledgments and Disclosure of Funding

This project is sponsored by CCF-Tencent Rhino-Bird Open Research Fund (No. CCF-Tencent RAGR20260112). The authors have no competing interests to report.

## References

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Iván Arcuschin, Jett Janiak, Robert Krzyzanowski, Senthooran Rajamanoharan, Neel Nanda, and Arthur Conmy. Chain-of-thought reasoning in the wild is not always faithful. *arXiv preprint arXiv:2503.08679*, 2025.

Darpan Aswal, Thomas Palmeira Ferraz, Yongxin Zhou, and Maxime Peyrard. Observable patterns are not explanations: A causal-geometric analysis of latent reasoning models. *arXiv preprint arXiv:2606.12689*, 2026.

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. Hopping too late: Exploring the limitations of large language models on multi-hop queries. *arXiv preprint arXiv:2406.12775*, 2024.

Jannik Brinkmann, Abhay Sheshadri, Victor Levoso, Paul Swoboda, and Christian Bartelt. A mechanistic analysis of a transformer trained on a symbolic multi-step reasoning task. *arXiv preprint arXiv:2402.11917*, 2024.

Vivien Cabannes, Charles Arnal, Wassim Bouaziz, Xingyu Yang, Francois Charton, and Julia Kempe. Iteration head: A mechanistic study of chain-of-thought. *Advances in Neural Information Processing Systems*, 37:109101–109122, 2024.

Shuochen Chang, Tong Bai, Xiaofeng Zhang, Qianli Ma, Qingyang Liu, Zhaohe Liao, Yibo Miao, and Li Niu. Unlocking the black box of latent reasoning: An interpretability-guided approach to intervention. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 34019–34032, 2026.

Xinghao Chen, Chak Tou Leong, Guo Wenjin, Jian Wang, Wenjie Li, and Xiaoyu Shen. What makes effective supervision in latent chain-of-thought: An information-theoretic analysis. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 15023–15042. PMLR, 06–11 Jul 2026. URL `https://proceedings.mlr.press/v306/chen26bi.html`.

Yingqian Cui, Zhenwei Dai, Bing He, Zhan Shi, Hui Liu, Rui Sun, Zhiji Liu, Yue Xing, Jiliang Tang, and Benoit Dumoulin. How do latent reasoning methods perform under weak and strong supervision? *arXiv preprint arXiv:2602.22441*, 2026.

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.

Connor Dilgren and Sarah Wiegreffe. Are latent reasoning models easily interpretable? *arXiv preprint arXiv:2604.04902*, 2026.

Subhabrata Dutta, Joykirat Singh, Soumen Chakrabarti, and Tanmoy Chakraborty. How to think step-by-step: A mechanistic understanding of chain-of-thought reasoning. *arXiv preprint arXiv:2402.18312*, 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.

Sayam Goyal, Brad Peters, María Emilia Granda, Akshath Vijayakumar Narmadha, Dharunish Yugeswardeenoo, Callum Stuart McDougall, Sean O'Brien, Ashwinee Panda, Kevin Zhu, and Cole Blondin. Scratchpad thinking: Alternation between storage and computation in latent reasoning models. In *NeurIPS 2025 Workshop on Regulatable ML*, 2025.

Halil Alperen Gozeten, M Emrullah Ildiz, Xuechen Zhang, Hrayr Harutyunyan, Ankit Singh Rawat, and Samet Oymak. Continuous chain of thought enables parallel exploration and reasoning. *arXiv preprint arXiv:2505.23648*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

Tianyu Guo, Hanlin Zhu, Ruiqi Zhang, Jiantao Jiao, Song Mei, Michael I Jordan, and Stuart Russell. How do llms perform two-hop reasoning in context? *arXiv preprint arXiv:2502.13913*, 2025b.

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.

Yifan Hou, Jiaoda Li, Yu Fei, Alessandro Stolfo, Wangchunshu Zhou, Guangtao Zeng, Antoine Bosselut, and Mrinmaya Sachan. Towards a mechanistic interpretation of multi-step reasoning capabilities of language models. *arXiv preprint arXiv:2310.14491*, 2023.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Tianjie Ju, Yijin Chen, Xinwei Yuan, Zhuosheng Zhang, Wei Du, Yubin Zheng, and Gongshen Liu. Investigating multi-hop factual shortcuts in knowledge editing of large language models. *arXiv preprint arXiv:2402.11900*, 2024.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Keito Kudo, Yoichi Aoki, Tatsuki Kuribayashi, Shusaku Sone, Masaya Taniguchi, Ana Brassard, Keisuke Sakaguchi, and Kentaro Inui. Think-to-talk or talk-to-think? when llms come up with an answer in multi-step reasoning. *arXiv preprint arXiv:2412.01113*, 2024.

Yingcong Li, Kartik Sreenivasan, Angeliki Giannou, Dimitris Papailiopoulos, and Samet Oymak. Dissecting chain-of-thought: Compositionality through in-context filtering and learning. *Advances in Neural Information Processing Systems*, 36:22021–22046, 2023.

Zhaoyi Li, Gangwei Jiang, Hong Xie, Linqi Song, Defu Lian, and Ying Wei. Understanding and patching compositional reasoning in llms. *arXiv preprint arXiv:2402.14328*, 2024.

Zirui Li, Xuefeng Bai, Kehai Chen, Yizhi Li, Jian Yang, Chenghua Lin, and Min Zhang. Dynamics within latent chain-of-thought: An empirical study of causal structure. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 65859–65883. PMLR, 06–11 Jul 2026. URL `https://proceedings.mlr.press/v306/li26c.html`.

Wenquan Lu, Yuechuan Yang, Kyle Lee, Yanshu Li, and Enqi Liu. Latent chain-of-thought? decoding the depth-recurrent transformer. *arXiv preprint arXiv:2507.02199*, 2025.

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

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

Neel Nanda and Joseph Bloom. Transformerlens. `https://github.com/TransformerLensOrg/TransformerLens`, 2022.

nostalgebraist. interpreting GPT: the logit lens. LessWrong, 2020. URL `https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens`.

Jingjie Qiu, Sheldon Yu, Xunyi Jiang, Jiajin Liu, Zihan Huang, Zhendong Chu, Tong Yu, Julian McAuley, and Junda Wu. Does latent chain-of-thought actually reason? a single, unifying taxonomy for latent reasoning. *arXiv preprint arXiv:2610.21182*, 2026.

Daking Rai and Ziyu Yao. 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.

Michael Rizvi-Martel, Guillaume Rabusseau, and Marius Mosbach. The illusion of superposition? a principled analysis of latent thinking in language models. *arXiv preprint arXiv:2604.06374*, 2026.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 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. *arXiv preprint arXiv:2502.21074*, 2025.

Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. *arXiv preprint arXiv:2409.12183*, 2024.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023.

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

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. *arXiv preprint arXiv:2405.15071*, 2024a.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. In *The Eleventh International Conference on Learning Representations*, 2023.

Yiming Wang, Pei Zhang, Baosong Yang, Derek F Wong, and Rui Wang. Latent space chain-of-embedding enables output-free llm self-evaluation. *arXiv preprint arXiv:2410.13640*, 2024b.

Yuhui Wang, Changjiang Li, Guangke Chen, Jiacheng Liang, and Ting Wang. Reasoning or retrieval? a study of answer attribution on large reasoning models. In *International Conference on Learning Representations*, volume 2026, pages 102235–102258, 2026.

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.

Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. SIM-CoT: Supervised implicit chain-of-thought. In *International Conference on Learning Representations*, volume 2026, pages 56721–56742, 2026.

Kevin Xu and Issei Sato. A formal comparison between chain-of-thought and latent thought. *arXiv preprint arXiv:2509.25239*, 2025.

Chenxiao Yang, Zhiyuan Li, and David Wipf. Chain-of-thought provably enables learning the (otherwise) unlearnable. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Sohee Yang, Nora Kassner, Elena Gribovskaya, Sebastian Riedel, and Mor Geva. Do large language models perform latent multi-hop reasoning without exploiting shortcuts? In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3971–3992, 2025b.

Zhipeng Yang, Junzhuo Li, Siyu Xia, and Xuming Hu. 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*, pages 22547–22575, 2025c.

Jiaran Ye, Zijun Yao, Zhidian Huang, Liangming Pan, Jinxin Liu, Yushi Bai, Amy Xin, Liu Weichuan, Xiaoyin Che, Lei Hou, et al. How do transformers learn implicit reasoning? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Yijiong Yu. Do llms really think step-by-step in implicit reasoning? *arXiv preprint arXiv:2411.15862*, 2024.

Zeping Yu, Yonatan Belinkov, and Sophia Ananiadou. Back attention: Understanding and enhancing multi-hop reasoning in large language models. *arXiv preprint arXiv:2502.10835*, 2025.

Jason Zhang and Scott Viteri. Uncovering latent chain of thought vectors in language models. *arXiv preprint arXiv:2409.14026*, 2024.

Yifan Zhang, Wenyu Du, Dongming Jin, Jie Fu, and Zhi Jin. Finite state automata inside transformers with chain-of-thought: A mechanistic study on state tracking. *arXiv preprint arXiv:2502.20129*, 2025a.

Yuyi Zhang, Boyu Tang, Tianjie Ju, Sufeng Duan, and Gongshen Liu. Do latent tokens think? a causal and adversarial analysis of chain-of-continuous-thought. *arXiv preprint arXiv:2512.21711*, 2025b.

Zhongwang Zhang, Pengxiao Lin, Zhiwei Wang, Yaoyu Zhang, and Zhi-Qin John Xu. Complexity control facilitates reasoning-based compositional generalization in transformers. *arXiv preprint arXiv:2501.08537*, 2025c.

Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *arXiv preprint arXiv:2505.12514*, 2025a.

Rui-Jie Zhu, Tianhao Peng, Tianhao Cheng, Xingwei Qu, Jinfa Huang, Dawei Zhu, Hao Wang, Kaiwen Xue, Xuanliang Zhang, Yong Shan, et al. A survey on latent reasoning. *arXiv preprint arXiv:2507.06203*, 2025b.

## Contents of the Appendix


**A Logit Lens** **16**

**B Attention Maps** **16**

**C Probing Intermediate Representations** **17**

C.1 Linear Probing Implementation Details . . . . . 18

**D Activation Patching** **19**

D.1 Activation Patching Implementation Details . . . . . 20

D.2 Visualization Details. . . . . 21

**E Training Configuration** **21**

**F Mechanistic Analysis on Three-hop Polynomial task.** **21**

**G Attention Knockout Analysis** **22**

**H State-Swap Intervention** **23**

**I Partial Latent Rollouts Concentrate on Late Intermediates for Longer Hops** **23**

**J Proofs for Theoretical Results in Section 5** **24**

J.1 Proof of Lemma 5.1: . . . . . 24

J.2 Proof of Lemma 5.2: . . . . . 24

**K Ablation Study** **24**

**L Comparing CODI-style latent CoT with CoT and Non-CoT Baselines** **25**

**M Finite-State Function Composition: Definition, Theory, Experiments, and Generality** **27**

M.1 Dataset Definition . . . . . 27

M.2 Theoretical Analysis . . . . . 28

M.3 Experimental Results . . . . . 30

M.4 Generality and Scope of FSC . . . . . 33

M.5 Implications for Latent-CoT Mechanisms . . . . . 34

**N Transition points depend on task difficulty** **34**

**O Extension to Coconut** **35**

**P Additional Related and Concurrent Work** **36**

**Q Compute Resources** **39**

**R Broad Impact** **39**

# 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

$$z_t \;=\; W_U \,\mathrm{LN}\Big(h_t^{(L)}\Big) + b_U, \qquad p(x_t = v \mid x_{<t}) \;=\; \mathrm{softmax}(z_t)_v, \tag{7}$$

where $W_U \in \mathbb{R}^{|V|\times d}$ is the unembedding matrix, $b_U$ is a bias, $\mathrm{LN}(\cdot)$ 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:

$$\tilde{z}_t^{(l)} \;=\; W_U \,\mathrm{LN}\Big(h_t^{(l)}\Big) + b_U, \qquad \tilde{p}_t^{(l)} \;=\; \mathrm{softmax}\Big(\tilde{z}_t^{(l)}\Big). \tag{8}$$

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 $s^\star$ (e.g., $s_t$ or the final $s_n$ in an $n$-hop instance), we measure the logit-lens probability assigned to $s^\star$, i.e., $\tilde{p}_t^{(l)}(s^\star)$. We plot $\tilde{p}_t^{(l)}(s^\star)$ (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

$$q_t = W_Q x_t, \qquad k_j = W_K x_j, \qquad v_j = W_V x_j, \tag{9}$$

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

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

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

$$\mathrm{Attn}(x)_t = \sum_j \alpha_{t\to j}\, v_j. \tag{11}$$

Stacking $\alpha_{t\to j}$ 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 $\alpha_{t\to j}$, 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 ($[\ell_1]$...$[\ell_6]$. 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 7 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, and Figure 8 shows that The `[Ans]` token places strong attention on the final input token $x_{n+1}$ across multiple n-hop tasks, indicating a copy-like pathway that routes $x_{n+1}$ directly into the answer readout.

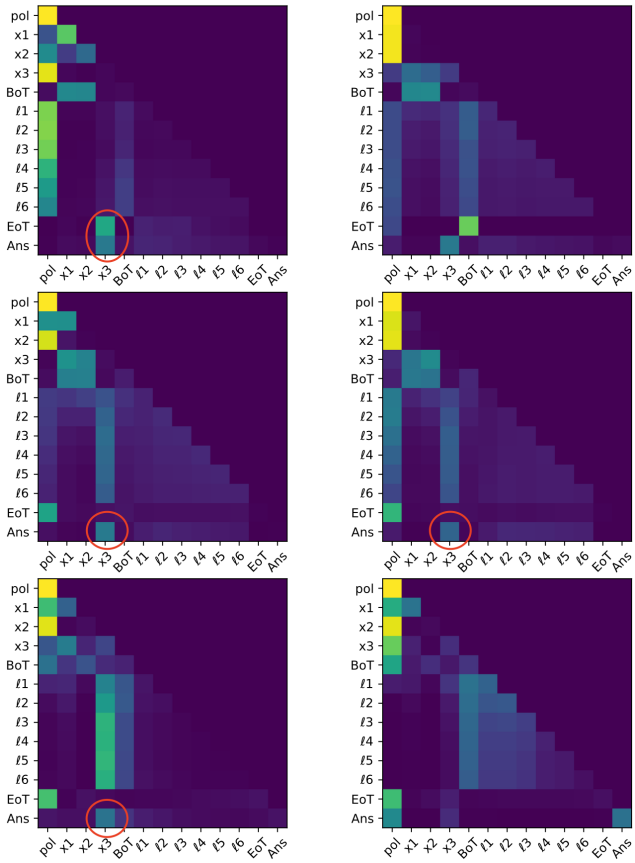
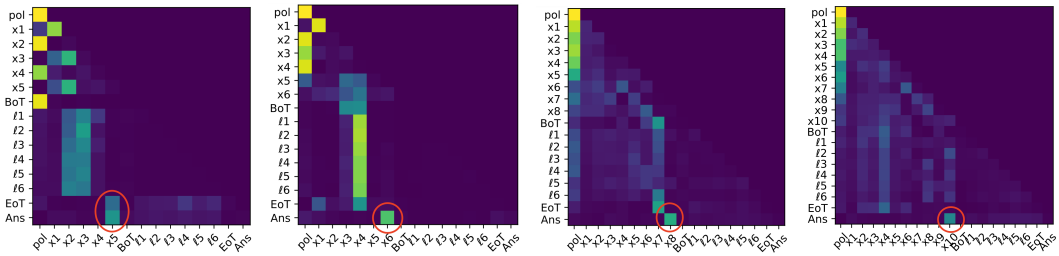



Figure 7: **Attention maps for the 3-layer, 2-head transformer on the two-hop polynomial task.** Rows correspond to layers (Layer 1 on top through Layer 3 on bottom), and columns correspond to heads (Head 1 on the left, Head 2 on the right). Across multiple heads, the `[Ans]` token places strong attention 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



(a) 4-hop task (b) 5-hop task (c) 7-hop task (d) 9-hop task

Figure 8: **Specialized attention head on the $n$-hop polynomial task**. The `[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 `[Ans]`.

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 $x_1, \ldots, x_{n+1}$. 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:

$$f(h) = Wh, \tag{12}$$

where $h \in \mathbb{R}^{256}$ is the hidden vector and $W \in \mathbb{R}^{50\times256}$. The 50 classes correspond to discrete values $\{0, 1, \ldots, 49\}$.

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

- **Input positions:** $x_i$ for $i \in \{1, \ldots, n+1\}$ (residual stream at each input token)
- **BoT (Beginning of Thought):** final encoder position before latent processing
- **Latent thought tokens:** $\ell_j$ for $j \in \{1, \ldots, 6\}$ (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:** $x_1, \ldots, x_n$ with $x_i \in \{1, \ldots, 49\}$
- **Intermediate states:** $s_1, \ldots, s_{n-1}$ with $s_i \in \{0, \ldots, 49\}$
- **Final answer:** `ans` $\in \{0, \ldots, 49\}$

**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 $N_{\text{correct}}$ samples.

**Train/validation/test splits.** We split $N_{\text{correct}}$ 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.

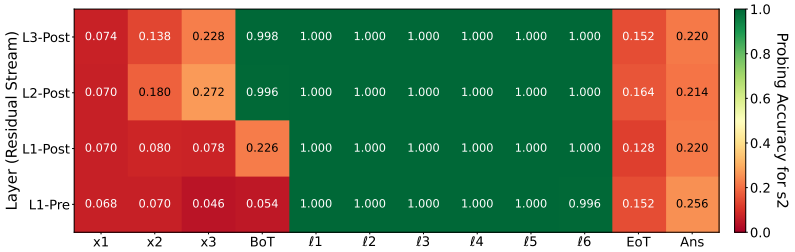



Figure 9: **Probing the intermediate state $s_2$ in the two-hop polynomial task.** Consistent with the logit-lens results, $s_2$ is readily decodable throughout the latent computation, indicating that the bridge state is formed before the model produces the final answer and supporting a sequential reasoning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

## 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.

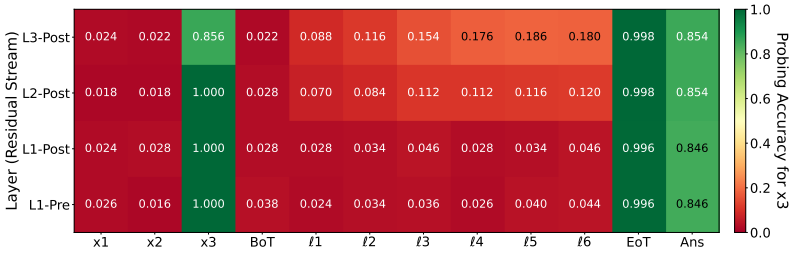



Figure 10: **Probing the input token $x_3$ in the two-hop polynomial task.** $x_3$ is strongly decodable at its own position and at the `[EoT]` and `[Ans]` tokens (probe confidence $\approx 1$ at $x_3$ and `[EoT]`, and $\approx 0.85$ at `[Ans]`). Consistent with the attention pattern in Figure 7, this supports a direct routing (copy-like) pathway from $x_3$ into the `[EoT]` and `[Ans]` residual streams.

### 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 $\mathbf{x} = (x_1, \ldots, x_n)$ 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 $\{1, \ldots, n\}$, latent tokens $\{\ell_1, \ldots, \ell_6\}$, 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 $\ell$, position/phase $p$):

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

We implement this via forward hooks in TransformerLens (Nanda and 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:

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

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

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

$$\text{Lift}_c = \frac{\text{Patched}_c - \text{Baseline}}{\text{Clean} - \text{Baseline}}. \tag{17}$$

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 ($\mathbb{P}(\hat{y} = y \mid \mathbf{x}) = 1$), ensuring clean runs provide a valid counterfactual. We corrupt each input position independently, running separate patching experiments for $x_1, x_2, \ldots, x_n$. 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 $3 \times 10^{-4}$ 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 $1, 2, \ldots, n+1$ (e.g., 4-hop training includes lengths 1–5). All models are trained in BF16 precision for computational efficiency.

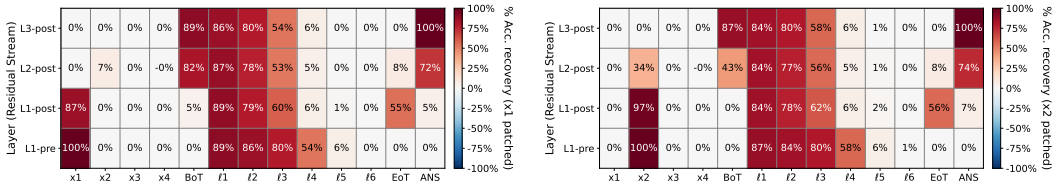
## F Mechanistic Analysis on Three-hop Polynomial task.

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.

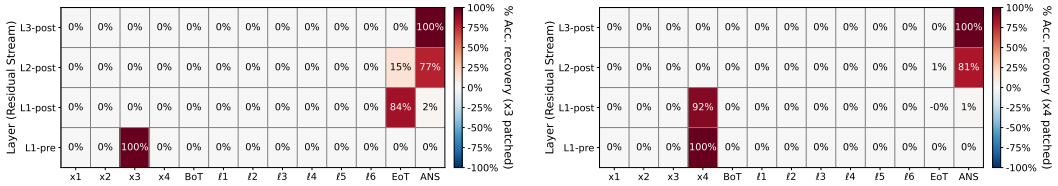
It is important to note that this step-by-step reasoning pattern does not emerge uniformly across all polynomial-task runs. In some cases, the model exhibits a *partial* latent trajectory: only $s_3$ becomes decodable in the latent steps, without clear evidence of $s_2$ formation. That said, across the different

random seeds and small model architectures we tested, the full step-by-step pattern appears reliably in the majority of settings.




(a) patching $x_1-$ corrupted run

(b) patching $x_2-$ corrupted run




(c) patching $x_3-$ corrupted run

(d) patching $x_4-$ corrupted run

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 $(\ell_1, \ell_2, \ell_3)$ 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.

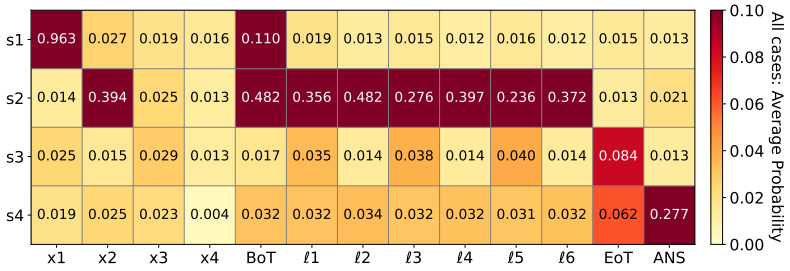



Figure 12: **Logit lens on intermediate states $s_1, s_2, s_3, s_4$ in the three-hop polynomial task.** For modulus $m = 50$, the logit lens shows that the bridge state $s_2$ becomes decodable early in the latent trajectory, indicating that it is formed and maintained in the latent channel. In contrast, $s_3$ becomes most decodable near the `[EoT]` boundary, suggesting that the next intermediate is computed or consolidated at the end of the latent segment. Each cell show average decoding probability across all layers and all test inputs.

## G Attention Knockout Analysis

In Section 4.2, to answer how the intermediate state s2 is routed to `[Ans]` token we conducted **targeted attention knockouts** to isolate the routing pathways. During the decoder phase, the `[Ans]` token attends over the full KV cache, which contains entries from the encoder tokens (`[POL]`, $x_1, \ldots, x_N$, `[BoT]`), latent tokens $(\ell_1, \ldots, \ell_K)$, and the `[EoT]` token. We selectively block `[Ans]` from attending to specific subsets of these KV positions by setting their pre-softmax attention scores to $-\infty$ at every decoder layer and head. This forces the softmax to assign exactly zero weight to those positions. We then measure accuracy on correctly predicted test examples (baseline = 100%).

We found that jointly knocking out `[Ans]` attention to `[BoT]`, all latent steps $\ell_1$–$\ell_6$, and `[EoT]` reduces accuracy to 3.8%, indicating that the information required for the final prediction is routed to `[Ans]` through this set of positions. To further isolate the routing pathway, we performed leave-one-in

ablations: we blocked attention to all of [BoT], $\ell_1$–$\ell_6$, and [EoT], while leaving exactly one position available at a time. Keeping [BoT] alone supports only 4.9% accuracy, showing that it is not a primary routing source. In contrast, recovery comes almost entirely from the early-to-middle latent steps: $\ell_1$ recovers 87% accuracy, $\ell_2$ 97%, $\ell_3$ 94%, $\ell_4$ 80%, $\ell_5$ 47%, and $\ell_6$ 28%, whereas [EoT] alone recovers only 4%. This experiment shows that the bridge-state signal $s_2$ reaches [Ans] primarily through the early-to-middle latent positions.

## H State-Swap Intervention

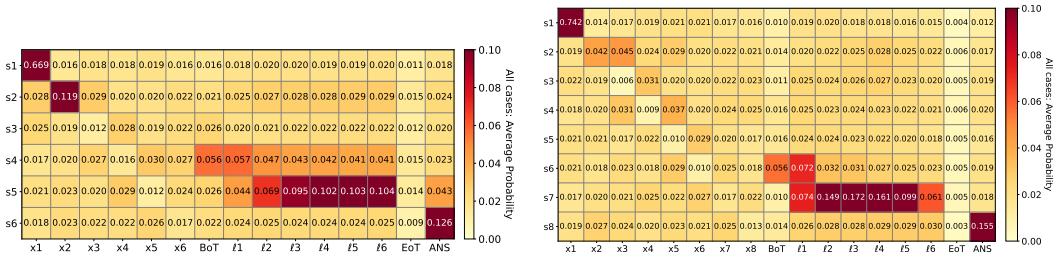
To further test whether the decoded bridge state $s_2$ in Section 4.2 is causally used rather than merely correlated with the output, we perform a state-swap intervention on sequence-length-3 examples. We construct suffix-matched prompt pairs $(A, B)$ that share the final input token $x_3$ but differ in the bridge state $s_2$ and therefore in the ground-truth answer $s_3$. At each latent position, we replace the residual stream of prompt $A$ with the cached activation from a clean forward pass of prompt $B$, and measure whether the model's prediction changes away from $A$'s answer and whether it flips specifically to $B$'s answer.

The intervention has a strong causal effect at the beginning of the latent trajectory. Swapping at [BoT], $l_1$, and $l_2$ changes the prediction away from $A$ in 98.7%, 97.5%, and 91.0% of cases, respectively, and flips it to $B$ in 90.1%, 80.1%, and 71.5% of cases. The effect then decays sharply: swaps at $l_3$ still affect about half of the examples, but by $l_5$ the flip-to-$B$ rate drops to 0.5%, and by $l_6$ and [EoT] it vanishes. This shows that the bridge-state representation is an active causal driver of the computation in [BoT] and the early latent steps, while later latent positions no longer serve as the main route by which $s_2$ determines the final answer.

## I 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.




(a) 5-hop task (b) 7-hop task

Figure 13: **Logit-lens evidence for a two-step partial latent rollout.** For longer-hop tasks (composite moduli), CODI often makes the last two intermediate states decodable within the latent-thought trajectory: $s_{n-1}$ appears earlier in the latent steps, followed by $s_n$ later.

## J Proofs for Theoretical Results in Section 5

**Prime and composite moduli.** A modulus $m \geq 2$ is *prime* if its only positive divisors are 1 and $m$ (e.g., $m = 41$), and *composite* otherwise (e.g., $m = 50$). For prime moduli, every nonzero multiplier is invertible, whereas composite moduli admit non-invertible multipliers that induce many-to-one state transitions. This distinction underlies the different compressibility properties analyzed in Section 5.

### J.1 Proof of Lemma 5.1:

**Lemma 5.1 [Bijection criterion]**: For $x \in R_m$, the map $f_x : R_m \to R_m$ 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 $\gcd(x, m) = 1$. □

### J.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 $\mu_x : R_m \to R_m$ by $\mu_x(s) = sx \pmod{m}$ and $f_x(s) = sx + b \pmod{m}$. Then $|\mathrm{im}(\mu_x)| = m/d$, and every fiber of $\mu_x$ has size $d$. Equivalently, $f_x$ is a $d$-to-1 map.

*Proof.* We view $R_m$ as an additive group. The map $\mu_x$ 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

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

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

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

so $|K| = d$.

For a group homomorphism $\varphi : G \to H$, each fiber $\varphi^{-1}(y)$ (for $y \in \mathrm{im}(\varphi)$) 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 $\mu_x$, every fiber has size $|K| = d$.

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

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

Finally, $f_x = \tau_b \circ \mu_x$ where $\tau_b(y) = y + b$ is a bijection on $R_m$. Therefore $f_x$ has the same fiber sizes as $\mu_x$, i.e., $f_x$ is also $d$-to-1. □

## K Ablation Study

For all subsequent ablations, we use the $n$-hop setting with $n = 31$, a 3-layer transformer with 2 attention heads per layer (unless stated otherwise), and modulus $m = 50$.

**Varying the number of latent steps.** We ablate the number of latent steps over $p \in \{1, 2, 3, 6, 9, 12, 20\}$. Across all values of $p$ tested, our main mechanistic observations remain mostly stable: on long $n$-hop tasks, the model consistently forms and propagates the *late* intermediate state—most reliably the final state $s_n$, and occasionally the last two states $(s_{n-1}, s_n)$.

To further investigate whether a larger latent-step budget can induce a more complete reasoning trajectory, we additionally evaluate 5-hop and 7-hop tasks using a 3-layer transformer with modulus $m = 50$. We vary the latent budget over $p \in \{3, 6, 12\}$ for the 5-hop task and $p \in \{4, 8, 16\}$ for the

7-hop task, including settings where the number of latent steps substantially exceeds the number of hops. Even with these expanded budgets, we consistently observe the same partial latent reasoning pattern: the model primarily represents the final one or two intermediate states rather than forming a complete step-by-step rollout.

These findings align with the analysis in Section 5: under composite moduli, contraction events make the label depend primarily on the terminal suffix, favoring late-bottleneck solutions. Accordingly, within this range, increasing $p$ does not reliably induce a longer internal rollout; the emergence of latent-step structure appears to be driven more by the task distribution (long-horizon updates under composite $m$) than by the available latent-step budget.

**Varying model depth and width.** We further test architectural robustness by sweeping depth and width, varying the number of layers in $\{2, 3, 4, 5, 6, 7\}$ and attention heads in $\{2, 4, 8\}$ (up to a 7-layer, 8-head student). Across these configurations, our mechanistic picture is mostly unchanged: on long $n$-hop tasks the model primarily represents and routes late intermediates—most reliably $s_n$, and sometimes $(s_{n-1}, s_n)$.

To further examine whether this behavior persists at larger scales, we additionally evaluate CODI using the GPT-2 Small architecture, comprising 12 layers, 12 attention heads per layer, and a residual-stream dimension of 768. For this experiment, we use a 9-hop polynomial task with modulus $m = 50$. Despite the substantial increase in model capacity, logit-lens analysis reveals that CODI continues to predominantly encode and propagate the final intermediate $s_9$ during its latent reasoning steps, consistent with the behavior observed in the 3-layer transformer.

This is consistent with Section 5: under composite moduli, frequent contractions make the label depend mainly on the terminal suffix, which naturally favors late-bottleneck solutions. Consequently, increasing model depth and width, even to GPT-2 Small scale, does not reliably elicit a deeper internal rollout; the observed latent-step behavior appears driven more by the task distribution than by the specific model configuration.

**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.

## L Comparing CODI-style latent CoT with CoT and Non-CoT Baselines

**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 $x_{i+1}$ (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

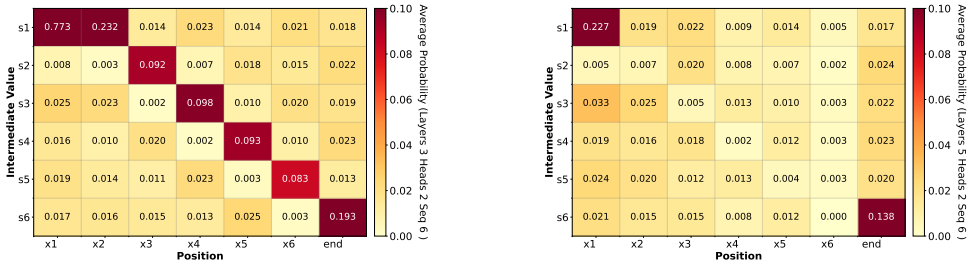
Table 1: **Accuracy (%) of CoT-trained models, a non-CoT standard transformer, and CODI across moduli $m$.** CODI and the non-CoT baseline degrade sharply when $m$ is prime. All results use a 3-layer, 2-head transformer with total sequence length 32, and report accuracy aggregated over all sequence lengths.

| 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 |

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 $s_1, \ldots, s_n$ across model scales and optimization settings (Cabannes et al., 2024). Consistent with this, Table 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.




(a) Layer-3, head-2.

(b) Layer-5, head-2.

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.

# M Finite-State Function Composition: Definition, Theory, Experiments, and Generality

This appendix gives the full definition, theoretical analysis, and experimental results for FSC, the finite-state function-composition task used in Section 6. The purpose of FSC is to test whether the compressibility effects observed in polynomial iteration are specific to modular arithmetic or instead reflect a more general property of sequential computation.

## M.1 Dataset Definition

**State space and function library.** Fix a finite state space

$$\mathcal{S} = \{0, 1, \ldots, N-1\}, \qquad |\mathcal{S}| = N.$$

FSC defines a fixed library of $K$ transition functions

$$\mathcal{F} = \{f_1, \ldots, f_K\}, \qquad f_i : \mathcal{S} \to \mathcal{S}.$$

The function library is constructed once, typically with a fixed random seed, and reused across all train and test examples. Thus the model is not asked to infer a fresh random function for each example; it must learn to use a fixed operation library.

**Bijective and compressive functions.** Let $\alpha \in [0, 1]$ denote the nominal fraction of bijective functions. Since $K$ is finite, we instantiate

$$K_{\text{bij}} = \text{round}(\alpha K), \qquad K_{\text{comp}} = K - K_{\text{bij}},$$

or another explicitly specified rounding rule. We define the realized bijective fraction as

$$\beta = \frac{K_{\text{bij}}}{K}.$$

This realized fraction $\beta$ may differ slightly from the nominal value $\alpha$.

For each of the $K_{\text{bij}}$ bijective functions, we sample a random permutation of $\mathcal{S}$. These functions have image size $N$ and preserve all state distinctions.

For each of the $K_{\text{comp}}$ compressive functions, we sample an image size

$$m_i \in \{m_{\min}, \ldots, m_{\max}\}, \qquad 1 \le m_{\min} \le m_i \le m_{\max} \le N - 1,$$

and construct $f_i$ so that

$$|\operatorname{Im}(f_i)| = m_i.$$

Smaller $m_i$ corresponds to stronger compression. We define the transition rank

$$r_i := |\operatorname{Im}(f_i)|.$$

Thus $r_i = N$ for bijections and $r_i = m_i < N$ for compressive functions. The quantity $N/r_i$ is the effective contraction factor of the transition.

**Implementation note.** For each compressive function, we first sample a target image size $m_i \in \{m_{\min}, \ldots, m_{\max}\}$ and then sample $m_i$ distinct output states from $\mathcal{S}$. The construction explicitly ensures that every selected output state appears at least once in the function table: we initialize the outputs with the selected image values, fill the remaining $N - m_i$ domain points by sampling from these values, and then shuffle the resulting list. Therefore each compressive function satisfies

$$|\operatorname{Im}(f_i)| = m_i$$

exactly.

**Example generation.** An example consists of an initial state and a sequence of operation indices:

$$s_0 \in \mathcal{S}, \qquad \sigma_1, \ldots, \sigma_T \in \{1, \ldots, K\}.$$

The hidden trajectory is generated by repeated function composition:

$$s_t = f_{\sigma_t}(s_{t-1}), \qquad t = 1, \ldots, T.$$

The model receives

$$(s_0, \sigma_1, \ldots, \sigma_T)$$

and predicts the trajectory

$$(s_1, \ldots, s_T).$$

**CODI-style serialization.** For explicit-CoT teacher training, we use the serialization

$$\texttt{[FSC]}\ s_0\ f_{\sigma_1}\ \cdots\ f_{\sigma_T}\ \texttt{[EoI]}\ s_0\ s_1\ \cdots\ s_{T-1}\ \texttt{[Ans]}\ s_T.$$

The student replaces the explicit intermediate chain with latent thought tokens:

$$\texttt{[FSC]}\ s_0\ f_{\sigma_1}\ \cdots\ f_{\sigma_T}\ \texttt{[BoT]}\ \ell_1\ \cdots\ \ell_p\ \texttt{[EoT]}\ \texttt{[Ans]}\ s_T.$$

This mirrors the polynomial-task setup: the teacher is trained to emit the full state trace, while the latent student must compress the computation into continuous thought tokens before predicting the final state.

### M.2 Theoretical Analysis

The polynomial analysis in Section 5 uses algebraic structure: over prime moduli, all nonzero multipliers are units and hence updates are bijections; over composite moduli, non-unit multipliers induce exact many-to-one contractions. FSC replaces this algebraic distinction with the transition-rank distinction $r_i = |\operatorname{Im}(f_i)|$.

**theorem 1** (Fully bijective FSC has non-contracting long-horizon dynamics)**.** *Suppose every function $f_i \in \mathcal{F}$ is bijective. Then for any operation sequence $\sigma_{1:T}$, the composed transition*

$$F_{\sigma_{1:T}} = f_{\sigma_T} \circ \cdots \circ f_{\sigma_1}$$

*is bijective. Hence, conditioned on $\sigma_{1:T}$, the final state $s_T$ preserves all distinctions between initial states.*

*Moreover, fix any step $t$ and compare two operation choices $i$ and $j$ at that step, holding the prefix and suffix fixed. Let $s_{t-1}$ be the prefix state before step $t$, and let $B_t$ be the suffix map after step $t$. Then the two final states satisfy*

$$B_t(f_i(s_{t-1})) = B_t(f_j(s_{t-1})) \quad \Longleftrightarrow \quad f_i(s_{t-1}) = f_j(s_{t-1}).$$

*Thus any distinction introduced by the operation choice at time $t$ is preserved by the later bijective suffix. In this sense, the fully bijective FSC regime has no intrinsic mechanism for erasing earlier state or operation distinctions.*

*Proof.* The first claim follows because a composition of bijections is a bijection. Therefore the map $s_0 \mapsto s_T$ is injective for every fixed operation sequence.

For the second claim, define

$$B_t = f_{\sigma_T} \circ \cdots \circ f_{\sigma_{t+1}}.$$

Since all functions are bijective, $B_t$ is bijective and hence injective. The final state after choosing operation $i$ at step $t$ is

$$y_i = B_t(f_i(s_{t-1})),$$

while the final state after choosing operation $j$ is

$$y_j = B_t(f_j(s_{t-1})).$$

Because $B_t$ is injective,

$$y_i = y_j \quad \Longleftrightarrow \quad f_i(s_{t-1}) = f_j(s_{t-1}).$$

Therefore, if the two candidate operations differ on the prefix state $s_{t-1}$, the final states must differ as well. Later bijective updates cannot erase the distinction. □

**Connection to the prime-modulus regime.** This theorem generalizes the no-contraction component of our prime-modulus analysis beyond modular arithmetic. In the polynomial task over a prime field, all nonzero multipliers are units, so every update $s \mapsto sx + b$ is a permutation and the recurrence never discards state information. The unrolled polynomial expression further shows that early inputs remain algebraically present in the final answer through multiplicative chains. FSC removes this algebraic structure, so there is no analogous closed-form expansion. Nevertheless, when $\alpha = 1$, the same information-preservation principle holds at the level of finite-state dynamics: every update is bijective, every suffix map is injective, and distinctions introduced at any earlier step cannot be merged by later updates. Thus, the fully bijective FSC regime serves as a non-algebraic counterpart to the prime-modulus regime: both lack many-to-one contractions, preserve long-range state distinctions, and therefore expose the fixed-budget limitations of latent-CoT models on incompressible sequential computation.

**Lemma 3** (Compressive updates induce rank contraction)**.** *Let $f_i : \mathcal{S} \to \mathcal{S}$ be a compressive FSC function with image size*

$$m_i = |\operatorname{Im}(f_i)| < N.$$

*Then $f_i$ maps all $N$ possible previous states into only $m_i$ possible next states. In particular, at least one output state has at least $\lceil N/m_i \rceil$ preimages. If $s$ is uniform over $\mathcal{S}$, then*

$$H(f_i(s)) \le \log m_i,$$

*so the update removes at least*

$$\log N - \log m_i = \log(N/m_i)$$

*bits of state capacity relative to a uniform $N$-state variable.*

*Proof.* The image of $f_i$ contains exactly $m_i$ states, so $f_i(s)$ is supported on at most $m_i$ values. Therefore its entropy is at most $\log m_i$. Since a uniform variable over $\mathcal{S}$ has entropy $\log N$, the update reduces the maximum state capacity by at least $\log(N/m_i)$. The preimage statement follows from the pigeonhole principle. □

**Interpretation.** The quantity $N/m_i$ is an effective contraction factor. This directly generalizes the $d = \gcd(x, m)$ contraction factor in the modular polynomial task, except that arbitrary FSC maps need not have equal-sized fibers.

**Lemma 4** (Lost distinctions cannot be restored by later deterministic maps)**.** *Let $r_i = |\operatorname{Im}(f_i)|$. For any subsequence of updates*

$$F_{a:b} := f_{\sigma_b} \circ \cdots \circ f_{\sigma_a},$$

*we have*

$$|\operatorname{Im}(F_{a:b})| \le \min_{a \le t \le b} r_{\sigma_t}.$$

*In particular, once a compressive function with image size $m_i$ occurs, the number of distinguishable prefix states after that point is at most $m_i$, and later deterministic updates cannot increase it.*

*Proof.* For any deterministic function $g$ and set $A$, we have $|g(A)| \le |A|$. After applying any transition $f_{\sigma_t}$, the reachable set has size at most $r_{\sigma_t}$. All later deterministic maps can only map this reachable set to another set of size no larger than its current size. Therefore the image size of the whole subsequence is bounded by the minimum transition rank encountered along the subsequence. □

**Interpretation.** This lemma is the formal basis for the late-bottleneck hypothesis. After a compressive update, the final answer can depend on the earlier prefix only through a compressed summary with at most $m_i$ states, plus the later suffix of updates.

**Lemma 5** (Terminal bijective suffix)**.** *Assume operation indices $\sigma_t$ are sampled independently and uniformly from the fixed library. Let*

$$\beta = \frac{K_{\text{bij}}}{K}$$

*be the realized fraction of bijective functions. Define the last compressive time*

$$\tau := \max\{t \le T : f_{\sigma_t} \text{ is compressive}\},$$

*with $\tau = 0$ if no compressive update occurs, and define the terminal all-bijective suffix length*

$$L := T - \tau.$$

*Then*

$$\Pr(L \ge k) = \beta^k,$$

*and, for $\beta < 1$,*

$$\mathbb{E}[L] = \sum_{k=1}^{T} \beta^k = \frac{\beta(1 - \beta^T)}{1 - \beta}.$$

*Proof.* The event $L \geq k$ occurs exactly when the final $k$ sampled operations are all bijective. Since operations are sampled independently and the probability of drawing a bijective operation is $\beta$, this event has probability $\beta^k$. The expectation follows from the tail-sum formula for a nonnegative integer-valued random variable:

$$\mathbb{E}[L] = \sum_{k=1}^{T} \Pr(L \geq k) = \sum_{k=1}^{T} \beta^k = \frac{\beta(1-\beta^T)}{1-\beta}.$$

□

**Interpretation.** When $\alpha = 1$, we have $\beta = 1$ and the whole sequence is bijective. When $\alpha < 1$, we have $\beta < 1$, and a recent compressive update appears with high probability. This makes the final state depend on a compressed prefix summary plus a short terminal suffix, paralleling the terminal all-unit suffix argument in the polynomial theory.

### M.3 Experimental Results

We next empirically test the predictions from Appendix M.2: compressive FSC regimes should make late-bottleneck latent reasoning viable, whereas fully bijective regimes should require sustained state propagation and therefore pose a harder setting for fixed-budget latent CoT.

We use a finite state space of size $N = 50$ and a fixed library of $K = 8$ transition functions with $m_{min} = 1$ and $m_{max} = 49$. We vary the fraction of bijective functions over

$$\alpha \in \{0.0, 0.1, 0.2, 0.6, 0.8, 1.0\}.$$

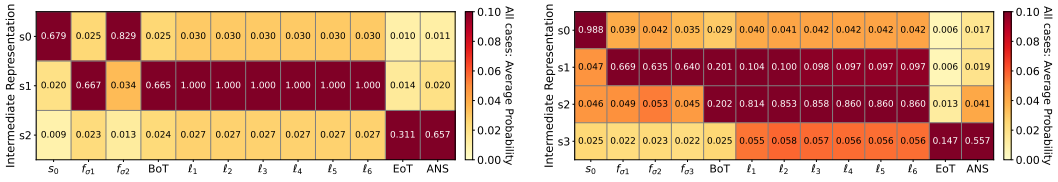
We further evaluate hop lengths

$$n \in \{2, 3, 4, 5, 7, 9\}$$

to compare short- and long-horizon behavior. When $\alpha = 1.0$, all transitions are bijective, yielding an incompressible regime analogous to the prime-modulus setting in the polynomial task. When $\alpha < 1.0$, the library contains at least one compressive transition, yielding a compressible regime analogous to the composite-modulus setting.

#### M.3.1 Compressive Region Where $\alpha < 1$

In the compressive regime, CODI exhibits the same short-horizon step-by-step pattern observed in the polynomial task. As shown in Figure 15, the logit-lens analysis reveals that the bridge state $s_1$ becomes decodable along the token/latent trajectory for 2-hop FSC tasks. For 3-hop tasks, both bridge states are recovered: $s_1$ is decodable from the early token/latent positions and remains accessible through the latent trajectory, while $s_2$ emerges within the latent channel. This pattern extends to the 4-hop setting, where we observe a full-latent rollout of the intermediate states. These results indicate that, when the horizon is short, CODI can form and maintain the ground-truth intermediate states required for faithful latent step-by-step computation.

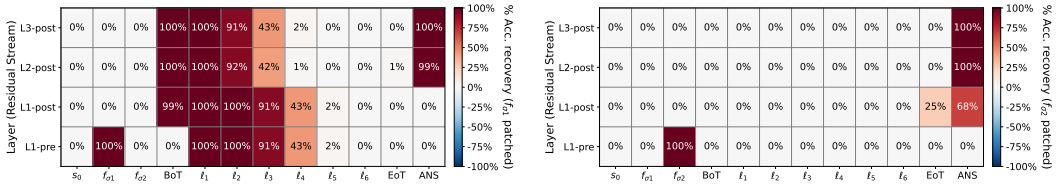



(a) 2-hop task and $\alpha = 0.2$ (b) 3-hop task and $\alpha = 0.6$

Figure 15: **Logit-lens analysis on short-horizon compressive FSC tasks.** For the 2-hop task in (a), the bridge state $s_1$ becomes decodable early in the latent trajectory when $\alpha = 0.2$, indicating that CODI forms the required intermediate state within at $f_{\sigma_1}$ token position and the latent channel. For the 3-hop task in (b), $s_1$ is already decodable at the $f_{\sigma_1}$ token position and remains decodable throughout the latent trajectory, while the next bridge state $s_2$ emerges within the latent channel. We observe qualitatively similar patterns across the tested values of $\alpha$ for 2-hop and 3-hop tasks, suggesting that CODI can realize faithful step-by-step latent reasoning in short-horizon FSC settings.

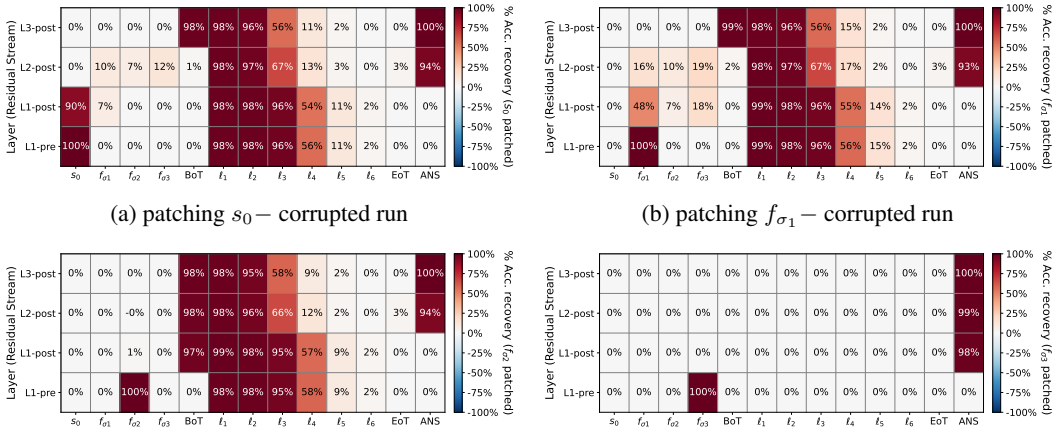
Activation patching further shows that the decoded bridge states are not merely correlational, but causally involved in the computation. On the two-hop FSC task, corrupting the first transition $f_{\sigma_1}$ disrupts the formation of the bridge state $s_1$. As shown in Figure 16a, patching clean activations into the latent-thought positions substantially restores accuracy, indicating that the latent channel stores information about $s_1$ that is necessary for the final prediction. In contrast, when the final transition $f_{\sigma_2}$ is corrupted, recovery concentrates at `[Ans]` rather than in the latent trajectory (Figure 16b). This mirrors the two-route mechanism observed in the polynomial task: the latent channel carries the intermediate state, while the final operation is routed more directly to the answer readout.

We observe the similar pattern on the three-hop FSC task. Corrupting the initial state $s_0$, the first transition $f_{\sigma_1}$, or the second transition $f_{\sigma_2}$ disrupts the computation of the later bridge state $s_2$. Patching clean activations into the latent-thought positions substantially restores accuracy, indicating that the latent channel carries information about $s_2$ that is causally necessary for the final prediction. By contrast, when the final transition $f_{\sigma_3}$ is corrupted, recovery again concentrates at `[Ans]` rather than in the latent trajectory. Together, these interventions show that short-horizon FSC tasks reproduce the same division of labor observed in the polynomial setting: latent thoughts compute and maintain bridge states, while the final transition is routed directly to the answer boundary. Details can be seen in Figure 17.




(a) $f_{\sigma_1}-$corrupted (b) $f_{\sigma_2}-$corrupted

Figure 16: **Activation patching for transition-function corruptions (two-hop).** Cells show mean accuracy recovery (%) over corrupted samples. When the first transition $f_{\sigma_1}$ is corrupted, patching clean activations into latent-thought positions restores performance, indicating that the latent channel carries the intermediate state $s_1$ needed for the downstream computation. In contrast, when the final transition $f_{\sigma_2}$ is corrupted, recovery is concentrated at `[Ans]`, consistent with a direct route that delivers the final operation to the answer readout.




(a) patching $s_0-$ corrupted run (b) patching $f_{\sigma_1}-$ corrupted run

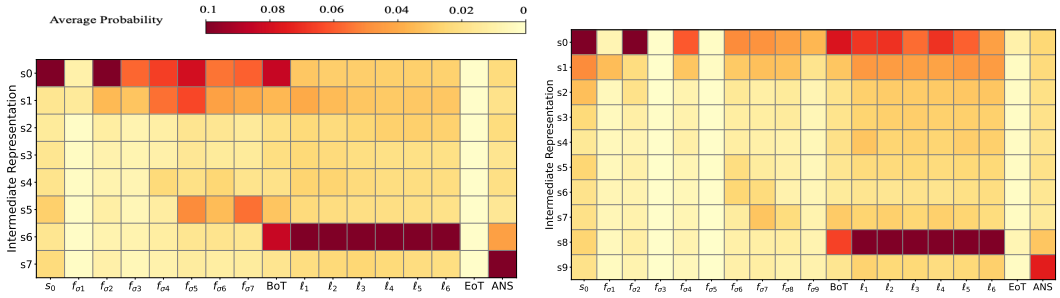
(c) patching $f_{\sigma_2}-$ corrupted run (d) patching $f_{\sigma_3}-$ corrupted run

Figure 17: **Activation patching for input-token corruptions in the three-hop FSC task.** When $s_0$, $f_{\sigma_1}$, or $f_{\sigma_2}$ are corrupted, patching clean activations into the latent-thought positions $(\ell_1, \ell_2, \ell_3)$ yields substantial accuracy recovery, indicating that the latent channel carries the intermediate information needed downstream. When $f_{\sigma_3}$ is corrupted, recovery localizes at `[Ans]`, consistent with a direct (copy-like) route that delivers $f_{\sigma_3}$ to the answer readout.

We defer discussion of the 5-hop task, which lies near the transition point between short- and long-horizon behavior, to Section N.

On longer-horizon FSC tasks, such as the 7-hop and 9-hop settings, we again observe a late-bottleneck latent reasoning pattern. As shown by the logit-lens analysis in Figure 18, the 7-hop task makes only the final bridge state $s_6$ readily decodable along the latent trajectory, while earlier intermediate states are largely collapsed. Similarly, in the 9-hop task, the latent trajectory primarily exposes the final bridge state $s_8$, with little evidence that the earlier states are explicitly maintained.

This behavior is consistent with the theoretical analysis in Appendix M.2: in compressive FSC regimes, low-rank transitions reduce the number of distinguishable prefix states, so the final state can often be determined from a compressed prefix summary together with a short terminal suffix. It also mirrors the mechanism observed in polynomial tasks with composite moduli, where non-unit multipliers induce exact algebraic contractions and CODI concentrates its latent computation on late intermediates rather than sustaining a full step-by-step rollout.




(a) 7-hop, $\alpha = 0.1$ (b) 9-hop, $\alpha = 0.6$

Figure 18: **Logit-lens analysis on long-horizon compressive FSC tasks.** For the $n$-hop FSC task, only the final bridge state $s_{n-1}$ becomes reliably decodable along the latent trajectory, while earlier intermediate states are largely collapsed. This late-intermediate pattern is qualitatively consistent across the tested compressive regimes with different values of $\alpha$.

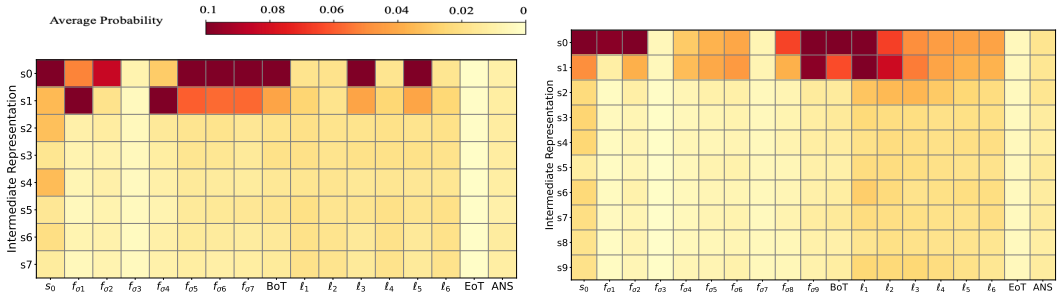
### M.3.2 Incompressible Region where $\alpha = 1$

**Long-horizon behavior in the incompressible FSC regime.** In the incompressible regime of FSC tasks, CODI exhibits step-by-step latent reasoning pattern for short-horizon instances such as 2-hop, 3-hop, and 4-hop FSC tasks. The logit-lens visualizations reveal a sequential emergence of the relevant bridge states along the latent trajectory, closely matching the behavior described above for compressive FSC tasks. This suggests that, when the reasoning horizon is short, CODI can faithfully construct the required intermediate states even when the task does not admit information compression. We again defer discussion of the 5-hop task, which lies near the transition point between short- and long-horizon behavior, to Section N.

However, this faithful step-by-step reasoning pattern breaks down for longer-horizon FSC tasks. For 7-hop and 9-hop instances, we observe that the latent reasoning trajectory collapses: late-stage intermediate states are no longer reliably formed or decodable. For example, in the 7-hop task, the final bridge state $s_6$ is no longer observable, and in the 9-hop task, the final bridge state $s_8$ similarly fails to emerge in the latent trajectory as seen in Figure 19. This contrasts with the compressive FSC regime, where late intermediates remain visible, indicating that compression helps preserve a tractable partial latent reasoning path over longer horizons.

This qualitative behavior is consistent with our theoretical analysis. In the incompressible FSC regime, corresponding to $\alpha = 1$, every transition is bijective, so the composed dynamics contain no many-to-one contraction that can erase state distinctions. Consequently, successful long-horizon prediction cannot exploit a compressed prefix summary induced by the task itself; it requires faithfully propagating the running state through the sequence of updates. This requirement mirrors the polynomial task over a prime modulus, where the algebraic structure prevents lossy compression of the intermediate computation and therefore necessitates a full intermediate reasoning path. Consequently, both settings impose a stringent information-preservation constraint: successful reasoning requires the model to retain and propagate all intermediate states across the entire horizon. Our empirical observation that CODI succeeds on short horizons but exhibits collapsed reasoning on longer horizons

is therefore aligned with the theory, which predicts that full-path reasoning becomes increasingly difficult as the horizon grows in non-compressive settings.




(a) 7-hop, $\alpha = 1.0$ (b) 9-hop, $\alpha = 1.0$

Figure 19: **Logit-lens analysis on long-horizon incompressible FSC tasks.** The final bridge state $s_{n-1}$ is no longer decodable along the latent trajectory, indicating a breakdown of step-by-step reasoning

### M.4 Generality and Scope of FSC

FSC is not intended to capture all sequential reasoning. Its rigorous scope is deterministic finite-state, operation-indexed sequential computation.

**Proposition 1** (Representation of finite-state operation-indexed tasks)**.** *Let $\mathcal{T}$ be any deterministic finite-state sequential task with finite hidden state space $\mathcal{Z}$, finite operation alphabet $\mathcal{A} = \{a_1, \ldots, a_K\}$, and Markovian update rule*

$$z_t = \delta(z_{t-1}, a_t).$$

*Then $\mathcal{T}$ can be represented as an FSC instance.*

*Proof.* Set $\mathcal{S} = \mathcal{Z}$. For each operation $a_i \in \mathcal{A}$, define

$$f_i(z) = \delta(z, a_i).$$

Given an operation sequence $a_{1:T}$, encode each operation by its index $\sigma_t$. Then the FSC recurrence

$$s_t = f_{\sigma_t}(s_{t-1})$$

exactly reproduces

$$z_t = \delta(z_{t-1}, a_t).$$

If $f_i$ is bijective, it belongs to the bijective part of the library; otherwise, its local compression level is captured by its transition rank $r_i = |\operatorname{Im}(f_i)|$. □

**Important special cases.** The following task families are direct special cases of FSC.

- **Finite automata.** A finite automaton has states $q_t$ and update rule $q_t = \delta(q_{t-1}, a_t)$. For each symbol $a$, define $f_a(q) = \delta(q, a)$.
- **Lookup-table transition systems.** A fixed library of discrete transition tables is exactly an FSC function library.
- **Finite-register update systems.** If the state consists of a tuple of finite registers, define $\mathcal{S}$ as the product of all register values. Each operation is then a deterministic function on the joint state.
- **Bounded finite-memory algorithms.** Any algorithm with a finite internal configuration under a fixed memory or length bound can be represented by encoding the full configuration as the FSC state. This may cause state-space blowup, but it is still a finite-state representation.

- **Modular polynomial iteration.** The polynomial task is

$$s_t = s_{t-1}x_t + b \pmod{M}.$$

For each multiplier $x$, define

$$f_x(s) = sx + b \pmod{M}.$$

The operation index $\sigma_t$ is the multiplier $x_t$. If $x$ is a unit modulo $M$, then $f_x$ is bijective. If $\gcd(x, M) = d > 1$, then $f_x$ is an exact $d$-to-one map with image size $M/d$. Thus the polynomial prime/composite distinction is a structured algebraic instance of the FSC bijective/compressive distinction.

**Scope of the FSC abstraction.** FSC is intended as a controlled abstraction for deterministic finite-state, operation-indexed sequential computation, rather than as a universal model of all sequential reasoning. This scope includes finite automata, lookup-table transition systems, bounded finite-register update systems,and the modular polynomial task studied in the main text. Tasks with unbounded memory, continuous or stochastic dynamics, or rich structured operations can be brought into this framework only after additional modeling choices, such as bounding the memory, discretizing the state space, encoding randomness as input, or expanding the state representation. Overall, FSC provides a clean setting for testing whether the compressibility mechanism observed in polynomial iteration persists beyond modular arithmetic.

### M.5 Implications for Latent-CoT Mechanisms

The FSC analysis suggests that the key variable in our polynomial experiments is not modular arithmetic itself, but transition-level compressibility. Fully bijective regimes preserve state distinctions and require full-history propagation, whereas compressive regimes contain low-rank updates that reduce the number of distinguishable prefix states and make late intermediates more predictive of the final answer. Our FSC experiments support this interpretation. On short-horizon tasks, CODI forms faithful bridge-state representations. On long-horizon compressive FSC tasks, it instead exhibits late-bottleneck partial rollouts, with late intermediates decodable and earlier states largely collapsed. Fully bijective FSC regimes remove this compression mechanism and therefore provide an incompressible control analogous to prime-modulus polynomial tasks. Thus, FSC shows that the polynomial task is a structured algebraic instance of a broader finite-state rank-contraction phenomenon.

**Practical implications.** Our findings suggest that sequential reasoning benchmarks should consider both task horizon and compressibility. In particular, bijective transitions preserve state distinctions, whereas many-to-one transitions enable compression-based shortcuts. Evaluating both dimensions can better assess whether latent-reasoning models perform faithful sequential computation.

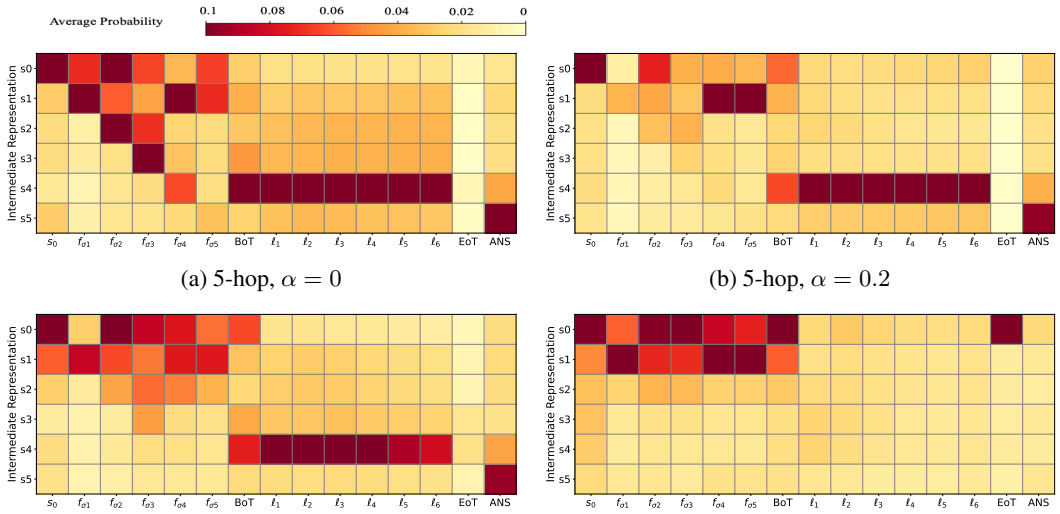
## N Transition points depend on task difficulty

We emphasize that the transition from faithful step-by-step latent reasoning to partial or collapsed reasoning is not tied to a universal hop length. Rather, it depends on the difficulty of the task distribution relative to the model and training budget. In our default polynomial setting with modulus $m = 50$, a 3-layer, 2-head CODI model trained with 2,500 examples per sequence length exhibits faithful bridge-state formation on 2-hop and 3-hop tasks (Figures 3 and 12), but shifts to a late-bottleneck partial rollout on tasks with 4 or more hops (Figure 5). Thus, under this setting, the transition appears around 4 hops. By contrast, under the same model and data budget but with prime modulus $m = 41$, the latent reasoning path collapses even for 2-hop tasks: we no longer observe a reliable faithful bridge-state trajectory.

The FSC experiments further support this view. With $N = 50$ states and $K = 8$ transition functions, the transition point varies with the bijective fraction $\alpha$, which controls the degree of compressibility. For example, on 5-hop FSC tasks, the fully compressive regime $\alpha = 0.0$ can still exhibit faithful step-by-step latent reasoning. In contrast, intermediate values such as $\alpha = 0.2, 0.6, 0.8$ shift toward late-bottleneck partial reasoning, while the fully bijective regime $\alpha = 1.0$ shows a collapse of the latent reasoning trajectory. Please see Figure 20 for more detail. This suggests that the transition point depends on the degree of compressibility: for $\alpha = 0.0$, faithful latent rollouts persist beyond 5

hops, whereas for intermediate values of $\alpha$ the transition to partial late-bottleneck reasoning already occurs around the 5-hop regime. This pattern is consistent with our theory: increasing $\alpha$ reduces the frequency of rank-contractive updates, making the task less compressible and requiring more sustained state propagation.

Overall, these results suggest that CODI-style latent reasoning reflects a capacity–difficulty tradeoff shaped by model capacity, training budget, and task structure. In the FSC setting, difficulty depends on the state-space size $N$, the number of transition functions $K$, the horizon length, the bijective fraction $\alpha$, and the compressive image-size range $[m_{\min}, m_{\max}]$. When the model has sufficient effective capacity, CODI can sustain faithful step-by-step latent computation. As the task becomes harder, it shifts toward late-bottleneck partial rollouts when compressive structure is available, but collapses when the transition system is effectively incompressible. Formalizing this capacity–difficulty tradeoff remains an important direction for future work.




(a) 5-hop, $\alpha = 0$

(b) 5-hop, $\alpha = 0.2$

(c) 5-hop, $\alpha = 0.6$

(d) 5-hop, $\alpha = 1.0$

Figure 20: **Transition point depends on FSC compressibility.** Logit-lens visualizations for 5-hop FSC tasks with $N = 50$ states and $K = 8$ transition functions, varying the bijective fraction $\alpha$. At the same horizon and model scale, CODI remains in a faithful step-by-step regime when $\alpha = 0.0$, shifts to a late-bottleneck partial-rollout regime for intermediate $\alpha$, and collapses in the fully bijective regime $\alpha = 1.0$. Thus, the hop length alone does not determine the transition from faithful reasoning to partial or collapsed reasoning; the transition point also depends on task compressibility, with larger $\alpha$ reducing rank-contractive updates and making sustained state propagation harder.

## O Extension to Coconut

**Training Configuration**. We provide additional training details for the Coconut experiments described in Section 8. We use a 3-layer GPT-2 model ($d_{\text{model}} = 128$, 2 attention heads) on the polynomial-iteration task defined in Section 3. For an $n$-hop task, the input consists of $n+1$ tokens $x_1, \ldots, x_{n+1}$, with $s_1 = x_1$ and $s_i = (s_{i-1}x_i + 1) \bmod m$ for $i = 2, \ldots, n+1$. The model predicts the final state $s_{n+1}$. The last intermediate state is $s_n$. Specifically, we adopt Coconut's curriculum-learning strategy to progressively replace explicit CoT steps with continuous latent thoughts. At curriculum stage $k \in \{0, \ldots, n\}$, the first $k$ CoT steps are replaced by latent thoughts, while the remaining $n - k$ steps receive explicit supervision. Each stage lasts 5 epochs, transitioning from fully explicit CoT to fully latent reasoning. We then train for an additional 25 epochs at the final curriculum stage, resulting in $5n + 30$ total epochs. Training uses AdamW with a learning rate of $3 \times 10^{-4}$ and a batch size of 128. We evaluate $n \in \{5, 6, 7\}$ under composite ($m = 50$) and prime ($m = 41$) moduli.

**Logit-Lens Visualization.** We provide additional logit-lens visualizations supporting the Coconut experiments presented in Section 8. Following the logit-lens methodology described in Appendix A,

we examine the decodability of ground-truth intermediate states across input-token positions, continuous latent-thought positions, and output-boundary positions. Each heatmap reports the average probability assigned to the corresponding ground-truth state, with rows representing intermediate states and columns representing token or latent-thought positions. The red dashed boxes highlight the decodability of the final intermediate state $s_n$ within the latent reasoning trajectory.

**Composite Modulus ($m = 50$).** Figure 21 presents the logit-lens results for $n \in \{5, 6, 7\}$. Across all three horizons, the final intermediate state $s_n$ exhibits the most consistent decodability within the latent thought positions. In contrast, several earlier intermediate states remain weakly decodable, and we do not observe a complete sequential emergence of all ground-truth intermediate states along the latent trajectory. Although Coconut exposes additional intermediate-state information in some settings, these results are consistent with a partial, late-bottleneck-like representational pattern, similar to the behavior observed in CODI.

**Prime Modulus ($m = 41$).** Figure 22 presents the corresponding results under the prime modulus. Unlike the composite-modulus setting, the final intermediate state $s_n$ is no longer consistently decodable from the latent thoughts across $n \in \{5, 6, 7\}$. The decoding probabilities for $s_n$ remain close to the uniform baseline, indicating that the late-intermediate reasoning signature observed under composite moduli largely disappears.

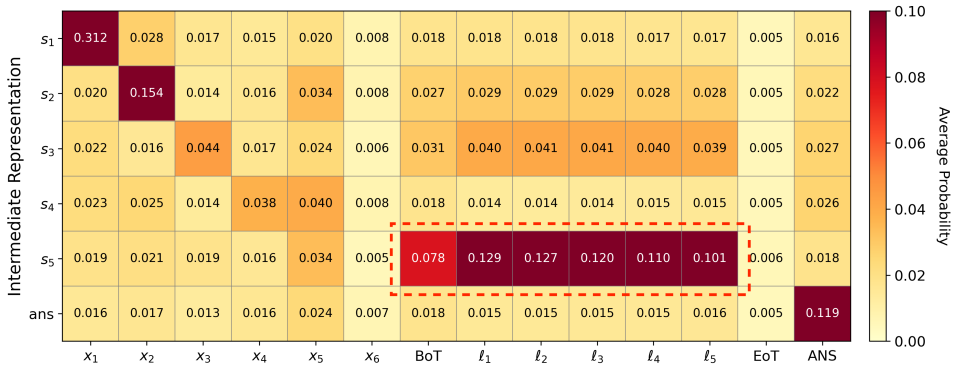
Taken together, these results provide additional evidence for the relationship between task compressibility and latent-reasoning behavior discussed in Section 5. Under composite moduli, many-to-one contractions make partial late-intermediate reasoning viable, whereas prime-modulus tasks lack this contraction-based structure. Despite differences in their training paradigms and intermediate-state representations, Coconut and CODI exhibit qualitatively similar patterns across these regimes.

# P Additional Related and Concurrent Work

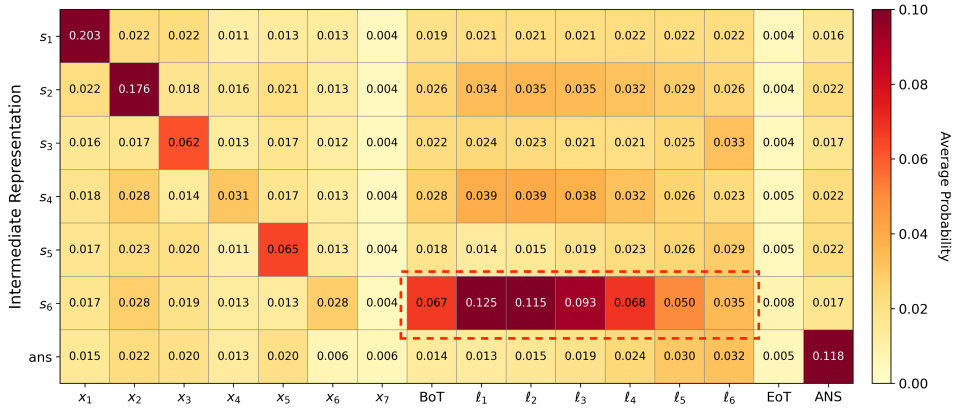
**Causal roles and faithfulness of latent steps.** Recent work has investigated whether continuous thought states actually contribute to a model's predictions, rather than merely encoding information that can be recovered by a probe. Li et al. (2026) model latent steps in Coconut and CODI as variables in a structural causal model and identify nonuniform causal roles and nonlocal dependencies across steps. Dilgren and Wiegreffe (2026) find that latent tokens are largely unnecessary on logical-reasoning tasks, whereas on arithmetic, where they are needed, gold reasoning traces can usually be decoded for correct predictions but much less often for incorrect ones. Cui et al. (2026) document shortcut behavior and show that the strength of latent-step supervision affects both reliance on latent computation and the diversity of represented hypotheses. More recently, Qiu et al. (2026) propose a taxonomy that separates what a latent chain represents from how the model uses it, testing whether the model uses the chain (via deletion and content replacement), what the chain encodes, and how its states evolve; they show that even an ordered chain can act as a placeholder. Relatedly, Wang et al. (2026) show that large reasoning models can rely on both CoT reasoning and memory retrieval when generating answers, highlighting the importance of distinguishing genuine reasoning from shortcut mechanisms.

**What can be inferred from latent representations?** Aswal et al. (2026) compare Coconut and CODI against controls that lack the proposed recurrence or curriculum, showing that recognizable latent patterns can occur without uniquely identifying the hypothesized reasoning mechanism; they advocate matched controls and causal interventions. Rizvi-Martel et al. (2026) find that only models trained from scratch show signs of superposition; in training-free and fine-tuned regimes it collapses or is bypassed via shortcuts, which they attribute to pretraining-induced token commitment and to model capacity. Complementarily, Chang et al. (2026) use structural and causal analyses to motivate interventions that improve latent reasoning at inference time. Our analysis likewise treats probing and logit-lens results as evidence of *decodability*, not by themselves as proof of computational use. On short-horizon sequential tasks, we pair these measurements with activation patching, state swaps, and attention knockouts to test the causal relevance and routing of bridge-state information. On longer tasks, the observed late-bottleneck patterns primarily characterize which ground-truth intermediate states are readily recoverable from the latent trajectory.

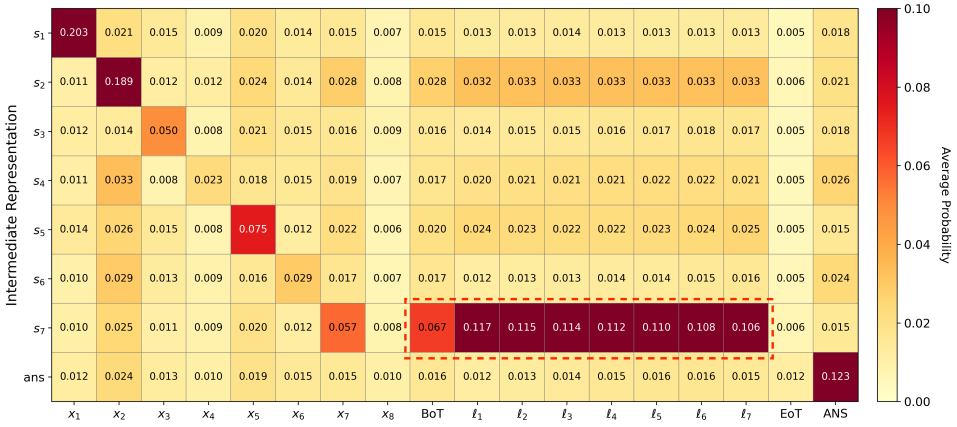
**Supervision and alternative latent-reasoning architectures.** Other studies address how latent-step organization can be encouraged during training. SIM-CoT (Wei et al., 2026) adds training-




(a) $n = 5$

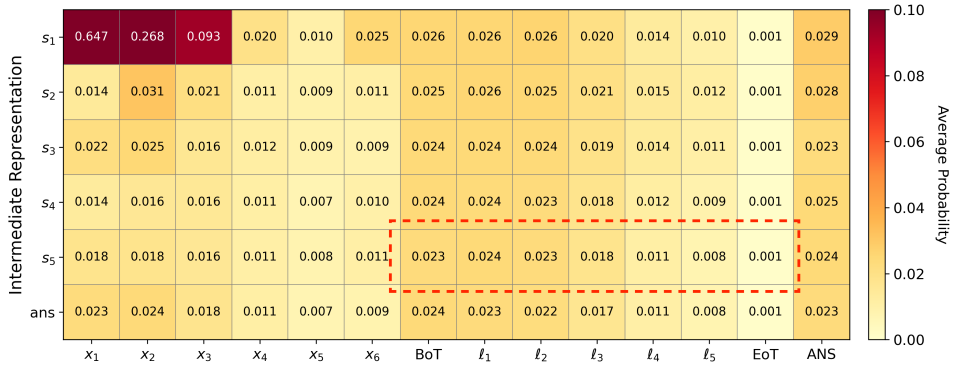



(b) $n = 6$

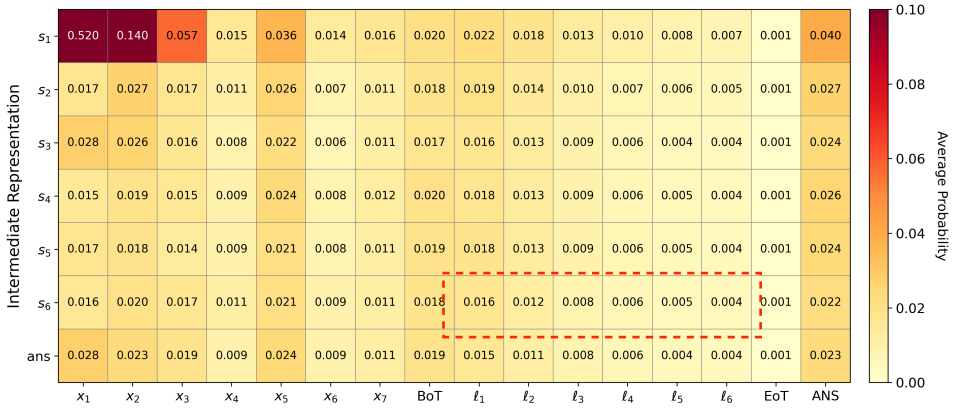



(c) $n = 7$

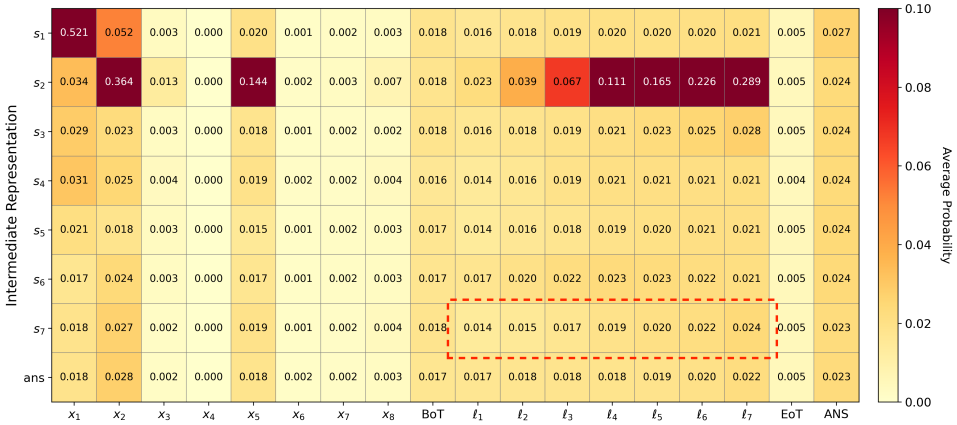
Figure 21: **Logit-lens analysis of Coconut on the $n$-hop polynomial task with modulus 50**. The final intermediate state $s_n$ is most consistently decodable along the latent trajectory, while earlier intermediate states are less consistently decodable, indicating a partial late-bottleneck-like pattern.




(a) $n = 5$




(b) $n = 6$




(c) $n = 7$

Figure 22: **Logit-lens analysis of Coconut on the $n$-hop polynomial task with modulus 41**. The final intermediate state $s_n$ (in the dotted rectangular boxes) is no longer reliably decodable along the latent trajectory, indicating a breakdown of the late-bottleneck-like pattern.

time step-level supervision and finds that it mitigates homogenization of implicit reasoning states. Chen et al. (2026) analyze gradient attenuation and representational drift, distinguishing trajectory supervision from supervision of the latent representation space. LOTUS (Fan et al., 2026) combines recurrent-depth computation with parallel supervision on latent reasoning positions, demonstrating another route toward CoT-aligned internal states. These results complement our loss ablations: in our CODI experiments, removing boundary-level feature distillation alone does not eliminate the late-intermediate signature, whereas removing the teacher objective as well does. This suggests that training objectives shape which computations are learned, but do not alone determine whether a task admits a compact successful solution.

**Distinct contribution of task-level compressibility.** Our central question is how the *structure of the sequential task* interacts with a limited latent-computation budget. In polynomial iteration, non-unit multipliers under composite moduli induce many-to-one transitions, while nonzero multipliers under prime moduli are bijective. Our finite-state function-composition (FSC) extension isolates the analogous distinction through transition rank. Rank-reducing updates merge previously distinguishable states, so the final answer can depend on earlier inputs through a compressed prefix summary; bijective updates do not provide this particular source of information loss. This notion of *task-level contraction* is different from the training-time homogenization or collapse of latent representations studied in some of the works above. By varying contraction structure while retaining known ground-truth trajectories, we characterize when CODI exhibits decodable step-by-step intermediates, partial late-intermediate trajectories, or a collapse of the corresponding decoding signature, and show in additional experiments that Coconut exhibits similar partial and collapsed patterns on longer horizons. We do not claim that compressibility uniquely determines a model's learned algorithm; rather, it changes which computational strategies are available and how difficult faithful recurrent computation may be under a fixed model and training budget.

## Q Compute Resources

All experiments were run on single-GPU workers without distributed training. Most training runs used one NVIDIA A100-SXM4-80GB GPU. The A100-80GB was used primarily for availability rather than necessity: the models studied in this paper are small GPT-2-style transformers trained on synthetic sequential tasks, and each run used only a small fraction of the available GPU memory. A typical 3-layer, 2-head CODI model with six latent tokens completed training within a few hours on a single A100 GPU. The mechanistic analyses were substantially cheaper than training. Logit-lens decoding, attention-map visualization, activation patching, state-swap interventions, and targeted attention knockouts were performed on trained checkpoints using forward passes over held-out examples. These analyses did not require distributed computation and used substantially less GPU memory than training; in practice, they can be reproduced on a single modern GPU. Linear probes were trained on frozen activations extracted from trained checkpoints and were also lightweight compared with model training. Overall, reproducing the main results does not require large-scale compute. Although our experiments were mostly executed on A100-SXM4-80GB GPUs, substantially smaller GPUs should be sufficient for the main training and interpretability experiments.

## R Broad Impact

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 and FSC 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.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes, the claims made in the abstract and introduction are validated through the proposed approach and experimental results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in the Conclusion section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Yes, please refer to Section 5; the full proof is provided in the Appendix. Moreover, additional theoretical results for the FSC task, along with their proofs, are provided in Appendix M.2.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Experiments are described in Section 4, and the detail of the experiments are described in the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code for reproducing the main experimental results is provided in the supplementary material.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Experiments are described in Section 4. Important setting can be found in the Appendix. The rest of the detail can be found in the code provided.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: The paper is primarily a mechanistic study rather than a performance-comparison study. The main claims concern qualitative and structural properties observed in particular trained model checkpoints, not statistically significant differences in aggregate benchmark performance. Most reported values are deterministic measurements on fixed checkpoints under fixed evaluation settings. Therefore, conventional error bars over repeated random seeds or statistical significance tests over performance metrics are not directly applicable to the central claims of the paper.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Yes, we included the computer resources information for experiments conducted in Section Q.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper conforms to NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We included the broader impacts statement in Appendix R.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All citations for assets used are given.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We have anonymized the code and described the code structure in the readme file at the time of the submission.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.