WHAT MAKES A GOOD LATENT THOUGHT FOR REASONING?

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent reasoning promises shorter generation by replacing token-level traces with compact continuous thoughts, but these representations must both preserve the reasoning trace and be reliably generated from the input. We ask which properties of latent thoughts support reliable reasoning under compression, using controlled permutation state-tracking tasks and varying compression and representation structure. Our findings are threefold. (i) At moderate compression, autoregressive (AR) generation remains highly accurate with far fewer emissions: 20 latent emissions recover a 160-token trace with near-perfect final-state accuracy, matching or exceeding a parameter-matched token-AR baseline with $8\times$ fewer emissions. (ii) At higher compression, reconstruction remains accurate while generation fails. Factorization or blockwise encoding restores accuracy, showing that latent structure strongly affects the ability to generate. With further compression, errors persist even with correct prefixes and on a simpler affine composition task, revealing a composition-capacity limit on how many state transitions the generator can realize within one emission. (iii) Iterative refinement is not a universal remedy. Diffusion does not recover highly compressed composition. On context-dependent graph walking, however, compression reverses the generator ordering: AR falls from perfect to chance, while diffusion improves. Together, these results show that usable compression depends on both latent structure and the computation packed into each emission. Effective latent reasoning therefore requires the right computational granularity for the representation, generator, and task.

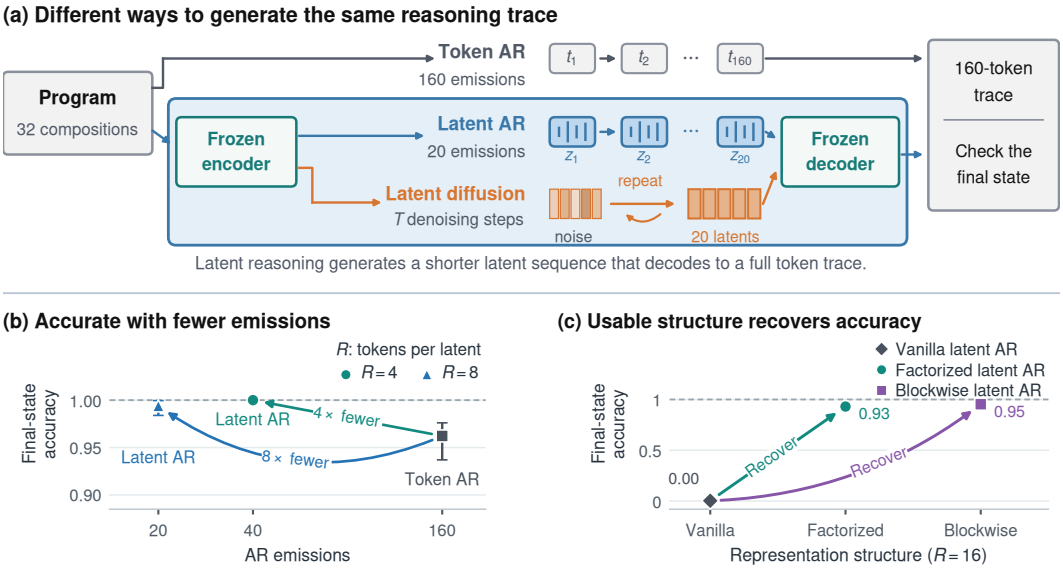

Figure 1: **Good latent thoughts combine compactness and usable structure.** **(a)** Token AR generates tokens directly. Latent AR and continuous diffusion share a frozen prompt encoder and trace decoder. **(b)** Latent AR stays accurate with 4–8$\times$ fewer emissions than parameter-matched token AR. **(c)** At fixed R=16, factorization and blockwise coding recover accuracy relative to vanilla latent AR with a global, entangled latent representation.

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

1 INTRODUCTION

Language models often reason through an explicit chain of thought (CoT). Intermediate tokens provide additional computation (Merrill & Sabharwal, 2024; Feng et al., 2023), but long traces require many sequential decoding steps. Latent reasoning aims to shorten these traces by carrying intermediate information in continuous vectors (Hao et al., 2025; Cheng & Van Durme, 2024; Tan et al., 2025). Realizing this promise requires understanding which latent representations support reliable reasoning. Compression changes more than sequence length. When several reasoning tokens share one latent, the generator must predict a representation carrying more of the computation in each emission. A representation may retain enough information to recover a solution yet be difficult to produce from the problem. This tension motivates our central question: *how should latent thoughts be structured to shorten reasoning traces without sacrificing reliable generation?*

Consider a program that repeatedly permutes five symbols, starting from $0, 1, 2, 3, 4$. Swapping 0 and 1 produces $1, 0, 2, 3, 4$, and the trace records the state after each update. Since each state contains five tokens, a 32-step program produces a 160-token trace. At a compression ratio R=8, every eight trace tokens are represented by one continuous vector, yielding 20 *latent thoughts*, or *latents* (Figure 2). Compression thus reduces the number of emissions while increasing the amount of reasoning represented by each one. We use this permutation state-tracking task over S_5 as our primary testbed because it lets us vary reasoning depth and compression independently while evaluating states exactly. Iterated permutation composition over S_5 is NC^1-complete (Barrington, 1989), making it a principled testbed for studying the computational limits of latent reasoning.

To separate reconstruction from generation, we train an encoder–decoder (codec) on correct traces and freeze it. We then train an autoregressive (AR) model to generate the latent sequence from the prompt and decode its predictions into a trace for evaluation. We vary compression, latent structure, and codec context to determine when generation fails despite successful reconstruction, and whether other representations recover accuracy. This setup reveals two limits to latent compression and shows that the relative strengths of AR and diffusion as the generator depend on the reasoning task.

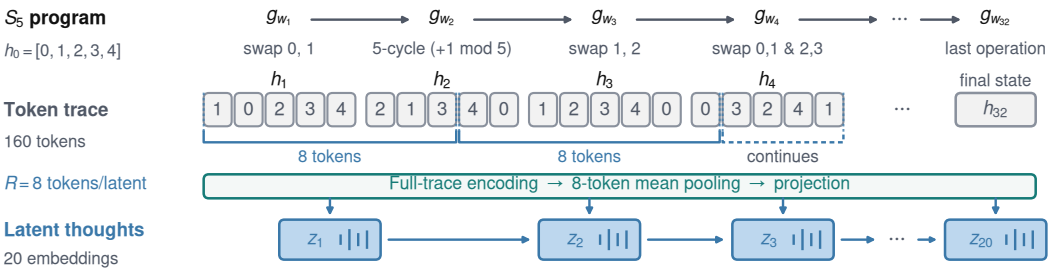

Figure 2: **Latent reasoning on an S_5 permutation program.** At R=8, a 32-step, 160-token reasoning trace is represented by 20 continuous latent thoughts, reducing the number of sequential emissions while preserving the underlying state trajectory.

At moderate compression, AR remains highly accurate with four to eight times fewer emissions than a parameter-matched token-level model. As compression increases, however, generation fails before reconstruction does: gold-trace latents still decode to correct answers, while predicted latents do not. Reorganizing the same latent budget through factorization or block-local encoding recovers accuracy at the same compression level (Figure 1), showing that the first bottleneck arises from how information is organized within the latent representation. At higher compression, this structural rescue no longer suffices. Errors persist even when the model is given correct prefixes, ruling out simple error accumulation, and also appear on a simpler affine task with a TC^0-computable closed form. Together, these results point to a second bottleneck: a limit on how many sequential state updates the trained generator can realize within a single emission. One might expect diffusion to offset this limit by repeatedly refining the full latent sequence. It does not: additional denoising fails to reliably recover composition accuracy. The picture changes on context-dependent graph walking. There, compression reverses the generator ordering: at greater reasoning depth, packing several retrieved transitions into one latent drives AR from perfect to chance while improving diffusion. Together, these results suggest that the difficulty of compression depends not only on how many reasoning steps are packed into a latent, but on what computation those steps require.

Our main contribution is to separate what a latent representation can encode from what a reasoning model can reliably generate. We show that the usable compression limit is governed jointly by representation structure, the generator's computation per emission, and the computation required by the task. Reorganizing the same latent budget can substantially extend this limit, whereas additional iterative computation does not universally do so, and the generation procedure matters instead when each latent must carry several context-dependent retrievals. These results recast latent-thought design as a problem of matching representation, generator, and task computation.

2 Experimental design

2.1 Primary task and exact evaluation

We use permutation state tracking over S_5 (Task 1): the input specifies a sequence of operations on five elements, and the output records the state after each operation. We vary the number of operations and the compression ratio independently, and evaluate the final state by exact match.

Task 1: Permutation state tracking (S_5). Program: starting from the identity, apply D permutations drawn uniformly from a fixed four-element generating set of S_5 (two transpositions, a 5-cycle, and a double transposition). **Output:** the full sequence of intermediate permutation states. **Evaluation:** exact match of the final state against the oracle. **Role:** a controlled composition task for the primary study of compression, representation structure, and generation.

Chain length versus compression. Each trace state uses five tokens (Figure 2). The *chain length* D counts permutation updates, distinct from model depth. We generate the full trace so that compression directly controls the number of AR emissions. Predicting only the five-token final state would leave little room to vary this budget, and a token-level AR control without intermediate traces fails on longer programs (Appendix D.1). The NC^1-completeness of iterated permutation composition motivates S_5 as a principled testbed for multi-step reasoning, but this asymptotic result does not predict where learned models fail in finite settings (Appendix A).

2.2 Codec stage: compression and code structure

We train a separate β-VAE encoder–decoder for each compression ratio R. The encoder contextualizes the full trace and pools every R tokens into one latent vector, while a non-autoregressive decoder reconstructs the full trace from the latent sequence alone. Because each state contains five tokens, each latent covers $R/5$ states on average. We refer to this quantity as *states per latent* of an AR emission. This is only a nominal measure: latent boundaries need not align with state boundaries, and contextualization allows information to mix across blocks. Importantly, codec compression and generation are separate design choices. Grouping tokens into codec blocks does not prescribe how latents are generated. After training, we freeze the codec and the latent standardization. We retain only codec configurations whose deterministic encode–decode reconstruction achieves 95% accuracy on gold traces. For each retained codec, we report its *reconstruction ceiling*: the final-state accuracy obtained by encoding and decoding the gold trace, without any generation error. Codec training and validation details are provided in Appendix C.2, and Figure 3 summarizes the overall training and evaluation pipeline.

2.3 Generator stage: representations and controls

We consider two different generators that operate in the same latent space defined by the frozen codec. Given the prompt, latent AR uses a causal transformer to generate one latent at a time, with a MAR-style rectified-flow head (Li et al., 2024a). Importantly, flow sampling refines only the newly generated latent; the backbone representation of the prefix remains fixed. Continuous latent diffusion, by contrast, repeatedly refines the entire latent sequence using a DiT backbone (Peebles & Xie, 2023) and rectified flow (Lipman et al., 2023). This gives the model additional iterative computation without adding latent positions, letting us test whether refinement can offset the computational burden of stronger compression. Both models use the same codec, data, and training budget. Parameter-matched and alternative-conditioning controls are given in Appendix D.2.

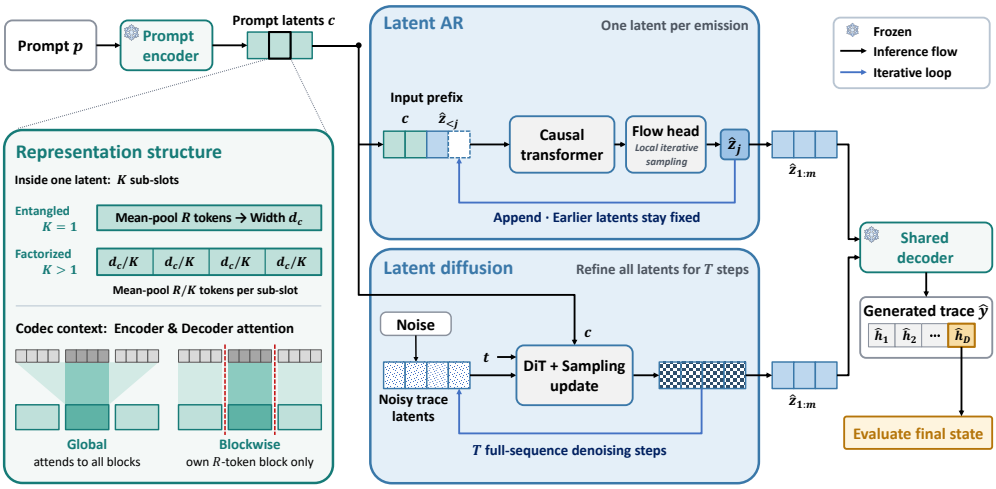

Figure 3: **Controlled framework for latent reasoning.** The encoded prompt conditions two generation procedures over the same latent representation: latent AR emits one latent at a time, whereas latent diffusion repeatedly refines the full latent sequence. We vary latent structure through factorization (K) and codec context (global or blockwise), then decode both outputs with the same frozen decoder and evaluate the final state exactly.

2.4 Test-time generation and protocol

At test time, both models receive only the prompt and generate the full latent sequence (Algorithm 2), which the shared frozen decoder maps back to a token trace. Evaluation uses independently sampled programs from the same distribution used to train both the codec and the generators. We do not specifically hold out particular states or token blocks. An independent oracle recomputes the correct answer from the prompt and compares it with the decoded final state, yielding *final-state accuracy*. For generation cost, we report the number of latent emissions m for AR and the number of full-sequence denoising steps T for diffusion. These counts reflect the number of sequential generation steps and are not equal to FLOPs or latency. For the token-level comparison in Figure 4, we use two controls: a 20-layer token AR model matched to latent AR in total parameter count, and a 16-layer token AR model matched to the latent AR backbone depth. We train them on the same task with the same number of optimization steps and the same training distribution over chain lengths. Table 2 summarizes the experimental grid, and Algorithm 1 gives the full training procedure.

3 Results

To understand when latent thoughts support reliable reasoning and what makes them do so, we address five research questions. **RQ1:** When does compression help reasoning? **RQ2:** What makes a latent thought easier to generate? **RQ3:** What limits further compression? **RQ4:** Can diffusion overcome the composition limit? **RQ5:** How does latent compression affect context-dependent retrieval?

3.1 (RQ1) When does latent compression help reasoning?

Direct comparison with token AR. At D=32, latent AR reaches 1.00 final-state accuracy at R=4 with 40 emissions and 0.994 at R=8 with 20, compared with 0.962 for a parameter-matched token AR model emitting all 160 tokens (Figure 1b, Figure 4a). All models use the same programs, trace targets, 25k generator updates, batch size, optimizer, learning rate, and chain-length distribution. The 20-layer token model matches latent AR's parameter count including its flow head; the backbone-matched 16-layer token control is also strong (0.990). Thus, the main benefit is accurate reasoning with fewer emissions. Codec pretraining remains an additional resource (Table 3).

Fewer emissions may reduce error compounding. Because autoregressive generation conditions on its own previous outputs, we track whether the entire decoded prefix remains correct. Let $S(k) = P(\hat{h}_1 = h_1, \ldots, \hat{h}_k = h_k)$ denote the probability that all states through k are correct. Figure 4b

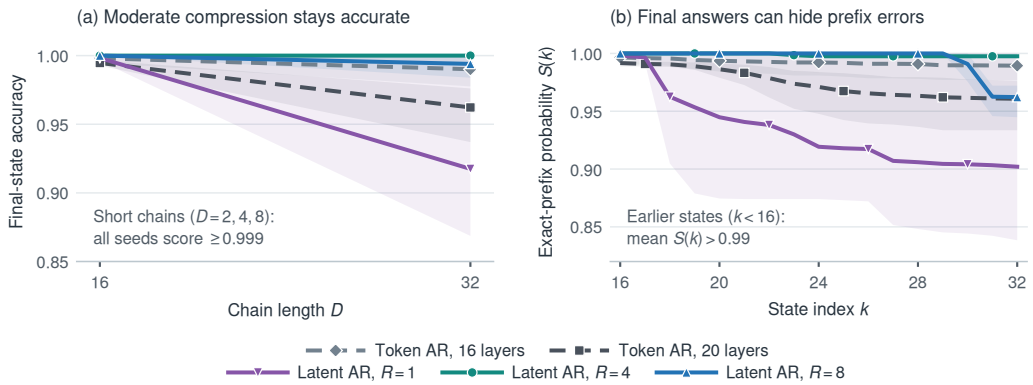

Figure 4: **Moderate latent compression preserves accurate reasoning with fewer emissions. (a)** Final-state accuracy at $D{=}16, 32$. **(b)** Exact-prefix probability at $D{=}32$, zoomed to $k{\geq}16$. Dashed lines denote token AR; solid lines denote latent AR. Lines and bands show three-seed means and min–max ranges on 2,048 shared programs per chain length. Full results are in Table 3.

compares $S(k)$ across the matched token and latent AR models. A trace through state k contains $5k$ tokens, corresponding to approximately $5k/R$ latent-AR emissions at compression ratio R. By shortening the autoregressive chain, compression reduces how much prefix errors can accumulate, which may explain the higher prefix survival of compressed models (Dziri et al., 2023). However, a deeper parameter-comparable $R{=}1$ AR model also improves prefix survival without shortening the trace (Appendix D.2), showing that shorter generation alone does not explain the difference. For later intervention comparisons, we summarize the decay of $S(k)$ with a fitted *prefix-survival factor* p, using $S(k) \approx c\,p^k$. Values closer to one indicate slower decay (Appendix E.2).

Finding 1. *Moderate compression maintains token-level reasoning accuracy while substantially shortening generation.*

3.2 (RQ2) What makes a latent thought easier to generate?

Generation fails before reconstruction, but restructuring the latent recovers. At $R{=}16$, latent AR accuracy drops sharply at $D{=}16$ and $D{=}32$, even though the frozen codec still reconstructs the correct final state from the ground-truth trace with 100% accuracy. Reconstruction also remains robust to moderate perturbations of the encoded latents (Appendix E.1), indicating that the failure is not simply due to information loss or a brittle decoder. The difficulty instead lies in generating the required latent sequence. Changing how information is organized in the code largely removes this failure. Factorization divides each R-token span into K sub-blocks and represents them separately within the latent, reducing how strongly their information is entangled. Holding the overall compression and model scale fixed, increasing K from 1 to 4 raises final-state accuracy at $D{=}32$ from 0.00 to 0.93. A $5\times$ parameter scale-up of the entangled ($K{=}1$) latent-AR does not yield a comparable recovery (Figure 5a). Blockwise encoding instead keeps the entangled $K{=}1$ code but restricts encoder and decoder attention to each latent's own R-token span, preventing information from mixing across blocks. This also recovers accuracy, reaching 0.95 at $D{=}32$ (Appendix D.3).

Finding 2. *The same compression can fail or succeed depending on code structure. Factorization and blockwise encoding recover AR accuracy without adding emissions.*

3.3 (RQ3) What limits further compression?

Further compression reveals a composition-capacity limit. At $R{=}32, K{=}8$, AR remains accurate at $D{=}2$ but falls near chance at $D{=}16$ and $D{=}32$, with a fitted prefix-survival factor of $p = 0.77$ across three seeds (Appendix D.3). Compared with the successful $R{=}16, K{=}4$ setting, the sub-block size remains four tokens, but each latent now contains twice as many sub-blocks. The codec itself remains reliable under the tested perturbations (Appendix E.1).

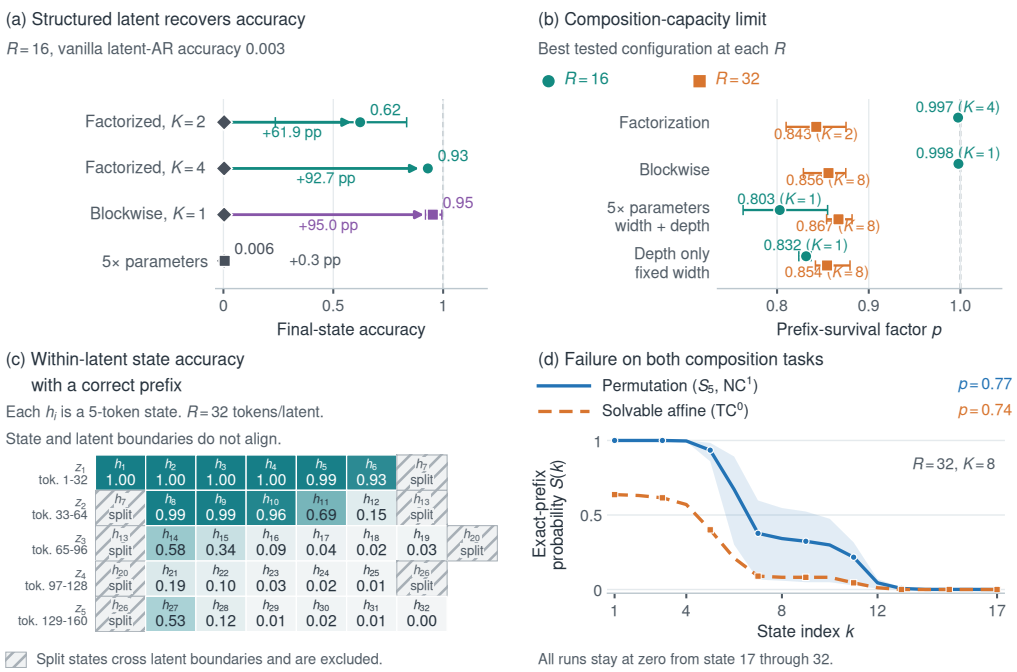

Figure 5: **Structure extends usable compression, but a generation bottleneck remains.** Latent AR at D=32. **(a)** R=16 accuracy gains (pp: percentage points). **(b)** Best three-seed mean per intervention axis and R. **(c)** Correct-prefix accuracy for fully contained states (R=32, K=8). **(d)** Prefix survival on permutation and affine tasks. Means and min–max ranges use three seeds, except the single-checkpoint panel (c) and affine contrast. Settings and probes: Table 6 and appendix E.2.

To determine whether the failure comes from earlier autoregressive errors, we predict each latent from the correct preceding latent sequence. At D=32, the 160-token trace is compressed into five latents with R=32. For each position, we predict that latent while keeping the rest of the sequence correct, then decode the resulting trace. Each state h_i occupies five tokens, and we score it only if all five lie within the predicted latent span. Even under this controlled setting, accuracy on these complete states is only 0.42 at K=8 (Figure 5c; Appendix E.2). Thus, errors arise within a single emission even when the preceding reasoning trajectory is correct.

Further changes to factorization, backbone depth, model scale, and codec locality improve prefix survival only partially, raising fitted p from 0.77 to at most 0.87, without recovering the high long-chain accuracy seen at lower compression (Figure 5b; Table 6). We therefore identify a second bottleneck, a *composition-capacity limit*: within the tested models and training budgets, the generator struggles to realize the larger number of consecutive state updates required by a single highly compressed emission.

The composition-capacity limit persists on a simpler task. To test whether the composition-capacity limit is specific to the algebraic complexity of S_5, we replace permutation composition with a simpler affine task over $\mathbb{Z}_5$. The task uses the same trace format and compression load, but its transformations admit a simple closed-form composition rule.

Task 2: Solvable affine composition. Program: compose D maps drawn uniformly from $\{x \mapsto x{+}1,\ x \mapsto x{+}2,\ x \mapsto -x\}$ over $\mathbb{Z}_5$. These maps generate the solvable group $\{x \mapsto \pm x + b\}$, whose composition has a TC^0-computable closed form. **Output:** the full sequence of intermediate permutation states, using the same five-token state format as Task 1. **Evaluation:** exact match of the final state against the oracle. **Role:** test the same sequential-composition load in a simpler algebraic setting.

The same high-compression failure remains. At R=32, K=8, the affine task has a fitted prefix-survival factor of $p = 0.74$, comparable to the S_5 setting (Figure 5d; Appendix D.4). Thus, the failure cannot be explained solely by the algebraic complexity of permutation composition. The ex-

istence of a closed-form solution does not imply that the trained model learns or uses that solution, so this experiment does not identify its internal algorithm. Instead, it shows that the observed generation bottleneck persists even when the underlying composition problem is substantially simpler.

Finding 3. *Simpler composition does not remove the high-compression failure. The bottleneck persists with correct prefixes and on an affine task with a closed-form solution.*

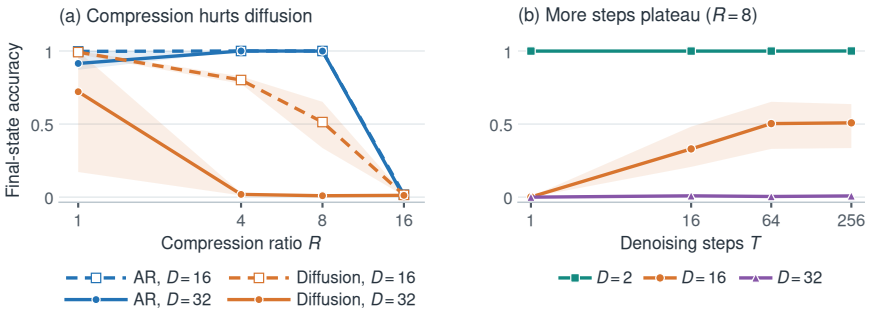

Figure 6: **Diffusion does not remove the compression failures. (a)** Latent AR and diffusion across compression ratios (diffusion uses the best T per seed). **(b)** Diffusion across step budgets at R=8. Lines and bands show three-seed means and min–max ranges.

3.4 (RQ4) Can diffusion overcome the composition limit?

Latent AR commits to each latent before moving to the next, whereas diffusion can repeatedly revise the full latent sequence without adding latent positions. If the high-compression failure reflects too much computation being packed into each latent, this additional refinement might help. We test this hypothesis on the same S_5 permutation state-tracking task, using the same frozen codec.

Latent diffusion does not remove the high-compression failure. With the global-context codec, diffusion degrades rapidly with both compression ratio and chain length (Figure 6a), including at R=4, 8, where latent AR remains accurate. Factorization improves diffusion at D=16, but accuracy remains near zero at D=32. Blockwise codecs likewise fail to recover (Table 8). This deficit is not explained by lossy prompt conditioning: providing the prompt without compression while compressing only the reasoning trace gives the same qualitative result (Figure 8a; Appendix D.2). It also persists under an alternative DiT conditioning architecture (Appendix D.2).

More denoising helps at intermediate chain lengths, then saturates. Increasing the number of denoising steps improves some intermediate settings (Figure 6b). At R=8, D=16 continues to improve through T=64 before plateauing below the reconstruction ceiling. In contrast, at D=32, diffusion final-state accuracy remains near zero as T increases. Thus, diffusion can provide useful additional computation, but it does not reliably compensate for the larger computational burden at high compression ratios and long chain lengths. This failure is not simply a consequence of giving diffusion fewer sequential updates. At D=32 and R=4, 8, latent AR reaches high accuracy with fewer sequential calls than diffusion, whose final-state accuracy remains near zero (Figure 8b). Increasing the number of sampling steps inside the AR flow head also does not rescue the high-load settings (Appendix D.6). Diffusion is therefore not sufficient to overcome the composition limit.

The continuous parameterization is an important part of the failure. When we discretize the same factorized representation with a post-hoc k-means codebook, masked diffusion recovers much of the lost accuracy. At R=16, K=4, D=32, masked diffusion reaches 0.64, compared with 0.01 for continuous latent diffusion and 0.68 for code-token AR (Appendix D.5). Alternative discrete samplers do not improve further, and the absorbing sampler stops changing once the schedule reveals at most one token per step (Proposition 2). These results show that the failure is not inherent to diffusion itself: how the latent is parameterized and refined strongly affects whether iteration is useful. Because discretization also changes the training objective, sampler, and effective computation, this comparison does not isolate discretization alone.

Finding 4. *Diffusion does not overcome the limit, but discreteness recovers much of the gap.*

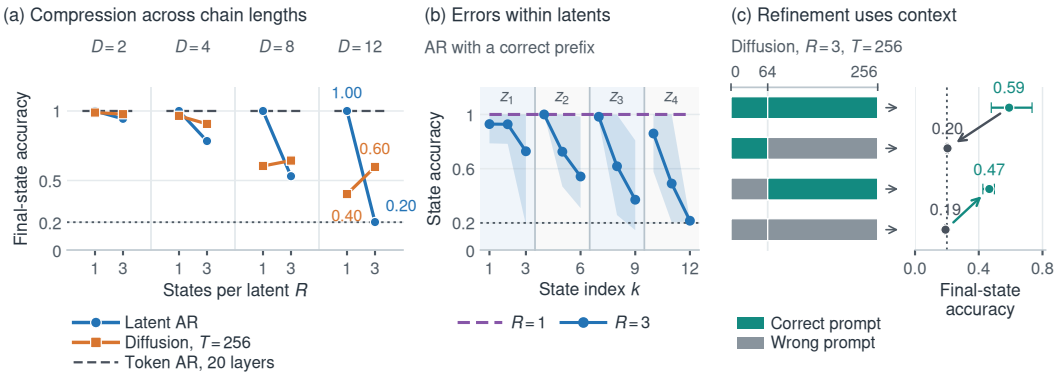

Figure 7: **Compression changes latent generation on graph walking. (a)** AR and diffusion under trace compression at $D{=}2, 4, 8, 12$. **(b)** AR state accuracy with ground-truth prefixes. Vertical divisions mark three-state $R{=}3$ latents. **(c)** Bars indicate whether the correct or wrong prompt is used before and after step 64 of 256. Arrows compare switching with keeping the initial prompt. All panels show three-seed means (Appendix D.4). Bands in (b) and error bars in (c) span seed min–max. Panels (b,c) use $D{=}12$. Dotted lines mark chance.

3.5 (RQ5) How does latent compression affect context-dependent retrieval?

The preceding results show that diffusion does not reliably recover accuracy on compressed composition traces. We now test whether compression behaves differently when each step must retrieve a transition from the prompt. In graph walking, the model must use the current node to look up the next transition from the graph given in the prompt. Packing several states into one latent therefore combines several dependent retrievals. This task lets us compare how causal generation and full-sequence refinement handle that burden.

Task 3: Context-dependent graph walking. Program: follow a deterministic walker through a layered graph, retrieving each transition from an edge-list context whose transition monoid contains S_5. **Output:** the sequence of nodes reached after each transition, with one token per node. **Evaluation:** exact match of the final node against the oracle. **Role:** replace fixed composition with context-dependent retrieval.

Compression reverses the generator ordering. Each node uses one token, so increasing R from one to three reduces a $D{=}12$ trace from twelve latents to four. At $D{=}12$, compression reduces latent-AR final-state accuracy from 1.00 to the chance rate of 0.20, while diffusion improves from 0.40 to 0.60 at fixed $T{=}256$ (Figure 7a). The task itself is not intrinsically difficult: both a parameter-matched token AR model and an $R{=}1$ latent AR model achieve 100% accuracy at every tested chain length across three seeds (Table 9). Without compression ($R{=}1$), AR remains exact across all tested chain lengths. Under compression ($R{=}3$), diffusion achieves higher mean accuracy than AR at every tested depth, and the gap widens on longer chains.

AR errors persist within individual latents. To test whether AR errors are simply inherited from earlier mistakes, we provide the correct latent prefix, predict only the next latent, and decode it while keeping the remaining latents fixed to their ground-truth values. At $D{=}12, R{=}3$, accuracy still drops within each latent. The exact pattern varies across seeds, but all three fall to near chance by the final state of the last latent, whereas $R{=}1$ remains exact throughout (Figure 7b). Thus, the collapse is not explained solely by earlier errors: AR also struggles to generate a single latent that must carry several dependent transitions (Appendix E.2).

Diffusion continues to use context during refinement. We test whether later denoising steps still rely on the graph prompt. At $D{=}12, R{=}3$, switching from the correct graph to an incorrect one after step 64 of 256 drives accuracy to chance, while switching back to the correct graph recovers much of the lost performance (Figure 7c; Appendix E.3). Thus, later refinement continues to use context and can correct an initially wrong trajectory. However, diffusion already outperforms AR after a single denoising step (0.50 vs. 0.20), so repeated context access cannot fully explain the gap (Figure 10a).

Finding 5. *Latent compression reverses the generator ordering on context-dependent retrieval. AR collapses as multiple retrieved transitions are packed into each latent, while diffusion improves.*

4 DISCUSSION

Latent thoughts are computational interfaces whose usefulness depends on both information preservation and ease of generation. Factorization and blockwise encoding make representations easier to generate at unchanged compression, but higher compression exposes the computational burden within each emission. This burden interacts with the task and generation procedure. On long permutation chains, high compression limits both AR and diffusion. The graph-walking results add an important qualification: this per-emission limit is not a single scalar capacity, but depends on the computation being packed into the latent. Compressing several context-dependent retrievals collapses AR while making diffusion comparatively stronger.

A good latent thought should therefore represent a unit of reasoning that is compact, structurally usable, and executable by the chosen generator. This argues against treating a fixed number of tokens per latent as the fundamental design variable. Real reasoning traces contain heterogeneous operations and their appropriate latent granularity may differ. Future latent reasoners may therefore benefit from adaptive boundaries, structured or partially discrete representations, and generation procedures selected around the computation carried by each latent rather than its token count alone. In this view, the goal is not to compress reasoning as aggressively as possible, but to discover a representation in which the model's computational steps and its latent steps are well matched.

Scope, limitations, and implications. The NC^1 structure of iterated permutation composition motivates S_5 as a principled testbed, but does not predict finite failure thresholds or behavior on other NC^1 tasks (Appendix A). Our codecs are trained separately and then frozen. Jointly learned representations are underexplored. Extending these findings to natural-language and agentic reasoning will require broader tasks and end-to-end measurements of accuracy and efficiency.

5 RELATED WORK

Latent reasoning and trace compression. Latent reasoning replaces token-level chains with shorter intermediate representations (Zhu et al., 2025b; Chen et al., 2025b). Continuous approaches reuse hidden states as latent thoughts (Hao et al., 2025; Zhu et al., 2025a) or compress spans of reasoning tokens into continuous codes (Cheng & Van Durme, 2024; Tan et al., 2025), while other methods use discrete latent tokens (Su et al., 2025). Other work learns adaptive or structured bottlenecks (Qu et al., 2026; Zhou et al., 2026; Massoli et al., 2026), with known failures on computation-heavy tasks (Zou et al., 2026; Li et al., 2026). We ask when a latent representation remains reconstructible yet becomes hard to generate.

Generation paradigms and iterative computation. Continuous latent thoughts can be generated autoregressively or through iterative refinement; LaDiR generates compressed latent trajectories by diffusion (Kang et al., 2026), while recurrent approaches include looped flows and Flow Reasoning Models (Suleymanzade et al., 2026; Helbling et al., 2026). Related token-space diffusion methods include Diffusion-of-Thought and LLaDA (Ye et al., 2024; Nie et al., 2025), and Vicentino (2026) compares AR and masked diffusion under matched budgets. Although masked diffusion is theoretically expressive (Svete & Sabharwal, 2026), it can behave nearly sequentially on harder tasks (Chen et al., 2025a). We study when iterative refinement helps under latent compression.

Expressivity and serial computation. Constant-depth log-precision transformers are restricted to uniform TC^0 (Merrill & Sabharwal, 2023; Strobl et al., 2024), whereas iterated products in non-solvable groups are NC^1-complete (Barrington, 1989). Serial reasoning can therefore expand effective expressivity (Merrill & Sabharwal, 2024; Feng et al., 2023; Li et al., 2024b), although learned models may exploit shortcuts (Liu et al., 2023).

6 CONCLUSION

Useful latent thoughts are not merely compact encodings of reasoning traces. They must also define units of computation that the generator can reliably execute. Moderate compression shortens AR generation without sacrificing accuracy, while structured codes extend this regime by making the same compressed information easier to generate. Beyond this regime, the limit depends on both the generator and the computation packed into each latent: diffusion does not resolve high-load composition, yet becomes comparatively stronger when a latent must carry several context-dependent retrievals. The goal of latent reasoning is therefore the right computational granularity, matching representation and generation to the computation each latent is asked to perform.

Reproducibility statement

All experiments run from one dependency-light repository. Data generators, oracles, and accounting use pure NumPy with exact-oracle unit tests. Models use PyTorch. Formal assumptions and proofs are in Appendix A; training objectives and algorithms are in Appendix B; data construction and model-selection protocols are in Appendix C; additional results and controls are in Appendix D. Including the complete intervention replication, the campaign to date used $\sim$133 H100-hours, $\sim$16 A100-hours, and $\sim$82 V100-hours. The code, including the pre-registered predictions, the follow-up contrasts, and the run artifacts, will be released with the paper.

AI use statement

We used generative AI tools to help with implementation and experiments, and to edit the manuscript for readability. All AI-assisted code was reviewed and tested by the authors, and all experimental results were validated against the saved run artifacts. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

Qiguang Chen, Hanjing Li, Libo Qin, Dengyun Peng, Jinhao Liu, Jiangyi Wang, Chengyue Wu, Xie Chen, Yantao Du, and Wanxiang Che. Beyond surface reasoning: Unveiling the true long chain-of-thought capacity of diffusion large language models. *arXiv preprint arXiv:2510.09544*, 2025a.

Xinghao Chen, Anhao Zhao, Heming Xia, Xuan Lu, Hanlin Wang, Yanjun Chen, Wei Zhang, Jian Wang, Wenjie Li, and Xiaoyu Shen. Reasoning beyond language: A comprehensive survey on latent chain-of-thought reasoning. *arXiv preprint arXiv:2505.16782*, 2025b. URL https://arxiv.org/abs/2505.16782. Accepted to Findings of the Association for Computational Linguistics: EMNLP 2026.

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*, 2024.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, 2023.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in neural information processing systems*, 36:70757–70798, 2023.

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. In *Conference on Language Modeling*, 2025. arXiv:2412.06769.

Alec Helbling, Andrey Bryutkin, Mauro Martino, Duen Horng Chau, Nima Dehmamy, and Hendrik Strobelt. Flow Reasoning Models: Turning flows into efficient recurrent reasoners. *arXiv preprint arXiv:2606.29150*, 2026.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamaki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. LaDiR: Latent diffusion enhances LLMs for text reasoning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=z5cPEZ4n6i.

Juncai Li, Ru Li, Yuxiang Zhou, Boxiang Ma, and Jeff Z. Pan. Chain of thought compression: A theoretical analysis. *arXiv preprint arXiv:2601.21576*, 2026.

Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. *Advances in Neural Information Processing Systems*, 37: 56424–56445, 2024a.

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

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *International Conference on Learning Representations (ICLR)*, 2023.

Fabio Valerio Massoli, Andrey Kuzmin, and Arash Behboodi. Reasoning as compression: Unifying budget forcing via the conditional information bottleneck. *arXiv preprint arXiv:2603.08462*, 2026.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations (ICLR)*, 2024.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/48b383b24230e0e6e649d9c98dae4d8c-Abstract-Conference.html.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Xingwei Qu, Shaowen Wang, Zihao Huang, Kai Hua, Fan Yin, Rui-Jie Zhu, Jundong Zhou, Qiyang Min, Zihao Wang, Yizhi Li, et al. Dynamic large concept models: Latent reasoning in an adaptive semantic space. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=vN7HfgJMMh.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality. In *International Conference on Machine Learning (ICML)*, 2025.

Yair Schiff, Subham Sekhar Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dalla-Torre, Bernardo P. de Almeida, Alexander Rush, Thomas Pierrot, and Volodymyr Kuleshov. Simple guidance mechanisms for discrete diffusion models. In *International Conference on Learning Representations*, 2025.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

DiJia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qinqing Zheng. Token assorted: Mixing latent and text tokens for improved language model reasoning. In *International Conference on Machine Learning*, 2025.

Ayhan Suleymanzade, Chanhyuk Lee, Floor Eijkelboom, Nicholas M. Boffi, İsmail İlkan Ceylan, and Jinwoo Kim. Thinking with looped flows. *arXiv preprint arXiv:2609.11801*, 2026.

Anej Svete and Ashish Sabharwal. On the reasoning abilities of masked diffusion language models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=BVnIsh4Nz1.

Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Ruihua Song, and Jian Luan. Think silently, think fast: Dynamic latent compression of LLM reasoning chains. In *Advances in Neural Information Processing Systems*, 2025.

Caio Vicentino. Autoregressive vs. masked diffusion language models: A controlled comparison. *arXiv preprint arXiv:2603.22075*, 2026.

Jiacheng Ye, Shansan Gong, Liheng Chen, Lin Zheng, Jiahui Gao, Han Shi, Chuan Wu, Xin Jiang, Zhenguo Li, Wei Bi, et al. Diffusion of thought: Chain-of-thought reasoning in diffusion language models. *Advances in Neural Information Processing Systems*, 37:105345–105374, 2024.

Cai Zhou, Chenxiao Yang, Yi Hu, Chenyu Wang, Chubin Zhang, Muhan Zhang, Lester Mackey, Tommi Jaakkola, Stephen Bates, and Dinghuai Zhang. Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner. In *International Conference on Machine Learning*, 2026. arXiv:2510.03206.

Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart J Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *Advances in Neural Information Processing Systems*, 38:79931–79963, 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.

Jiaxuan Zou, Yaozhong Xiong, and Yong Liu. Capabilities and fundamental limits of latent chain-of-thought. *arXiv preprint arXiv:2602.01148*, 2026.

APPENDIX CONTENTS

Appendix A Formal statements and proofs **14**
A.1 Composition and computational scope 14
A.2 Token-block contents and exposure 15
A.3 Absorbing-sampler step cap 16
Appendix B Training objectives and algorithms **16**
B.1 Continuous codec and objective 16
B.2 Two-stage training and generation 17
B.3 Discrete representations, objectives, and samplers 18
Appendix C Experimental setup and model selection **19**
C.1 Worked prompt and trace example 19
C.2 Codec training and validation 20
C.3 Generator training and model selection 21
C.4 Graph-walking task and serialization 21
Appendix D Additional results and controls **21**
D.1 Token–latent comparison and grid replication 21
D.2 Scale and conditioning controls 23
D.3 Interventions at the two compression limits 24
D.4 Task-family contrasts 27
D.5 Discrete objectives and sampler results 30
D.6 AR flow-head sampling control 31
Appendix E Mechanism probes **33**
E.1 Codec and grading audits 33
E.2 Chain and emission-localization probes 33
E.3 Repeated context use 35

A FORMAL STATEMENTS AND PROOFS

We first state the asymptotic motivation for permutation composition, then count possible local token blocks, and finally establish the step cap of the discrete absorbing sampler. These results do not predict the finite compression thresholds of the trained generators.

A.1 COMPOSITION AND COMPUTATIONAL SCOPE

Setting. For permutation composition, $G = S_5$ and the fixed generating set is $g_0 = (0\,1)$, $g_1 = (0\,1\,2\,3\,4)$, $g_2 = (1\,2)$, and $g_3 = (0\,1)(2\,3)$, with $k = 4$ generators. A program $w = (w_1, \ldots, w_D) \in \{0, \ldots, k-1\}^D$ produces states $h_0 = e$ and $h_i = g_{w_i} \circ h_{i-1}$: each new generator acts on the output of the current permutation. Each state is serialized into five tokens. Experiments pad the trace to $5D_{\max}$ tokens and group it into R-token blocks associated with latents. The complexity statement below instead considers programs of unbounded length, without a fixed padding limit. The sampler proposition concerns discrete code-token generation.

Proposition 1 (No depth-free composition)**.** *Let f be a deterministic transformer forward map, defined at every input length in the bounded-precision model of Merrill & Sabharwal (2023). Its layer count, width, and weights are fixed, its arithmetic precision is $O(\log n)$ bits at input length n, and its component computations satisfy that model's space bounds. We use logspace-uniform circuit classes throughout this proposition. Say that f* computes span c *if it returns $h_c = g_{w_c} \cdots g_{w_1} h_0$ for every initial state $h_0 \in G$ and program $w \in \{0, \ldots, k-1\}^c$. Define $\mathcal{S}(f) = \{c \geq 1 : f$ computes span $c\}$. If $\mathsf{TC}^0 \neq \mathsf{NC}^1$, then $\mathcal{S}(f) \neq \mathbb{N}$. In particular, the consecutive* composition budget

$$C(f) = \max\{c \in \mathbb{N}_0 : \{1, \ldots, c\} \subseteq \mathcal{S}(f)\}$$

is finite, with the condition at $c = 0$ understood vacuously.

Proof. Suppose $\mathcal{S}(f) = \mathbb{N}$. The constant-depth simulation of Merrill & Sabharwal (2023) places the family computed by f in TC^0. Fixing $h_0 = e$ and testing whether the constant-size output equals e would therefore solve the word problem over the four generators in TC^0.

The word problem asks whether a sequence of group elements multiplies to the identity. Over the full alphabet G, it is NC^1-complete under AC^0 reductions (Barrington, 1989). To reduce it to our generating set, choose a representing generator word for each of the 120 group elements. These words can all be made the same constant length: $g_0^2 = e$ and $g_1^5 = e$ give identity words of lengths two and five, and every integer at least four has the form $2a + 5b$ for nonnegative integers a, b. Choosing a sufficiently large common length thus permits padding every representative without changing its product. Replacing each input symbol with its fixed-length representative is an AC^0 reduction. Hence the assumed f would imply $\mathsf{NC}^1 \subseteq \mathsf{TC}^0$, contrary to the separation assumption. The first missing span $c_* = \min(\mathbb{N} \setminus \mathcal{S}(f))$ is therefore finite, and $C(f) = c_* - 1$. □

Scope. $C(f) + 1$ is the first span at which universal correctness fails, not a threshold beyond which every span fails. The proposition gives no numerical threshold or average-case error rate, and does not directly describe our bounded-context, stochastic generators. A fixed number of bounded-precision backbone, head, and decoder updates still admits a constant-depth circuit simulation, but this does not identify the source of measured errors. Likewise, $R/5$ is a nominal state count, not a measure of computation inside a latent vector (Lemma 1). Associativity permits an $O(\log D)$ parallel-prefix algorithm (Liu et al., 2023). The $\lceil 5D/R \rceil$ emissions needed for an unpadded trace follow from the output format, not a lower bound on composition. Nor does NC^1-completeness by itself transfer the observed behavior to other NC^1 tasks.

Why the affine contrast has a constant-depth solution. Write each affine operation as $f_i(x) = a_i x + b_i \pmod 5$, with $a_i \in \{-1, 1\}$. Left composition gives

$$f_D \circ \cdots \circ f_1(x) = \left(\prod_{i=1}^{D} a_i \right) x + \sum_{i=1}^{D} b_i \prod_{j=i+1}^{D} a_j \pmod 5.$$

Each suffix product depends only on the parity of the number of negative coefficients. These parities and the final sum modulo five can be computed by constant-depth, polynomial-size threshold

circuits. Applying the same formula to every prefix also computes the full trace in TC^0. This establishes the simpler complexity of this particular affine family, without assuming that a trained model learns the closed-form algorithm.

A.2 Token-block contents and exposure

Lemma 1 (Realizable block contents)**.** *Let a pad-free block of R trace tokens start at zero-based offset $o = 5q+r$, where $0 \le r < 5$ is its* phase *and $o+R \le 5D$. It intersects $s_r = \lfloor (r+R-1)/5 \rfloor + 1$ consecutive states $h_{q+1}, \ldots, h_{q+s_r}$, where $s_r \in \{\lceil R/5 \rceil, \lceil R/5 \rceil + 1\}$, and its content is a function of $(h_{q+1}, w_{q+2}, \ldots, w_{q+s_r})$. Hence the number of realizable pad-free contents at phase r is at most $|G|\, k^{s_r-1}$, against $|G|^{s_r}$ for unconstrained states. A sub-block of at most five tokens has at most $|G|$ contents if it lies inside one state and at most $|G|k$ if it straddles a state boundary.*

Proof. Token t belongs to state $\lfloor t/5 \rfloor + 1$, which gives s_r; the two values arise because $(r+R-1)/5$ and $(R-1)/5$ differ by less than one. Given h_{q+1}, each later state in the block is $h_i = g_{w_i} h_{i-1}$, so the block is determined by the anchor state and $s_r - 1$ generator indices. A sub-block inside one state shows entries of a single permutation, and one straddling a boundary shows entries of h_i and of $g_{w_{i+1}} h_i$, which are determined by (h_i, w_{i+1}). □

Token-block counts and expected exposure. For $R = 16$, every phase intersects four states, giving at most $120 \cdot 4^3 = 7{,}680$ contents per phase. This is over four orders of magnitude below the 120^4 combinations of four unconstrained full states. A four-token sub-block has at most 120 contents within a state or 480 across a boundary. For $R = 32$, the corresponding whole-block upper bound ranges from $120 \cdot 4^6 = 491{,}520$ to $120 \cdot 4^7 = 1{,}966{,}080$, depending on phase. On the order-10 affine group with three generators, the $R = 16$ upper bound is $10 \cdot 3^3 = 270$.

The base generator budget contains $25{,}000 \times 128$ examples, of which one fifth, or $N = 640{,}000$, have chain length 32. Under a uniform-content approximation, the $R = 16$ upper bound corresponds to about $N/7{,}680 \approx 83$ exposures per content at a fixed longest position. This is an expectation under a simplified sampling model, not a count of distinct latent vectors observed during training. Actual token-block probabilities depend on position and the random walk over G.

Observation 1 (Token-block exposure and code structure)**.** *Under the uniform-content approximation above, the counts suggest repeated exposure to $R = 16$ token blocks. Factorization applies a shared projection to smaller sub-blocks, potentially making shared structure easier to reuse. The monotone improvement with K supports this interpretation (Table 7), but does not establish a learned compositional rule.*

The encoder contextualizes the full trace before pooling, so identical local blocks can receive different latent representations. Token-block exposure therefore does not establish latent-space coverage. A blockwise codec, which makes each latent a function of its own block, recovers AR accuracy at R=16 without factorization (Appendix D.3), consistent with this latent-space reading. Both interventions show that the tested generator can handle this load with a different representation; Proposition 1 does not predict failure at this finite load. The affine contrast also limits the counting explanation: its smaller alphabet removes the short-chain failure, but long-chain performance remains below ceiling (Section 3.3).

A.3 ABSORBING-SAMPLER STEP CAP

Proposition 2 (The absorbing serial cap)**.** *Consider absorbing-state sampling of $n \geq 1$ generated positions in $T \geq 1$ steps with a denoiser F_θ that is a deterministic function of the current partially revealed sequence and fixed prompt. It receives neither a step index nor the total budget T. The initial sequence is entirely masked. At step $s \in \{0, \ldots, T-1\}$, a predetermined integer quota $q_s \geq 0$ of still-masked positions, no more than the number still masked, is revealed using the most confident predictions and their argmax values in the non-mask alphabet. Revealed positions are never rewritten. Confidence and argmax ties use fixed deterministic rules that do not depend on s or T. Then:*

(a) at most $\min(T, n)$ steps change the state. Calls at zero-quota steps can be omitted without changing the output;

(b) under the cumulative schedule $q_s = \rho_{s+1} - \rho_s$ with $\rho_s = \mathrm{round}(ns/T)$ (nearest-integer rounding, with ties to even), exactly $\min(T, n)$ steps change the state;

(c) under this schedule, for every $T \geq n$ the output equals that at $T = n$: one position is revealed per state-changing step, in confidence order.

Proof. (a) Let M_s be the set of masked positions after s steps. Then $M_{s+1} \subseteq M_s$ and $|M_0| = n$, and each state change removes at least one position, so at most n of the T steps change the state. A call at a zero-quota step leaves the state unchanged. Omitting it changes neither the later inputs nor the predetermined quotas, so determinism preserves all subsequent outputs. (b) ρ_s is nondecreasing with $\rho_0 = 0$ and $\rho_T = n$. For $T < n$ consecutive arguments differ by more than one, so $\rho_{s+1} \geq \rho_s + 1$ and every step reveals at least one position. For $T = n$, all quotas are exactly one. For $T > n$, consecutive arguments differ by less than one, so $q_s \in \{0, 1\}$ and exactly n steps have $q_s = 1$. (c) By (b), for $T \geq n$ each state-changing step reveals the single most confident masked position with its argmax value, so the state trajectory, and hence the sample, is the same for every $T \geq n$. □

Scope. Our absorbing sampler skips zero-quota calls, executing exactly $\min(T, n)$ calls under the cumulative schedule. Here $n = K\lceil 5D_{\max}/R \rceil = 40$ for $R = 16$, $K = 4$, and $D_{\max} = 32$. With deterministic evaluation and fixed tie breaking, all $T \geq 40$ on a fixed prompt execute the same sequence of 40 deterministic one-position update attempts and therefore give identical outputs. The implementation does not explicitly suppress the mask logit: if a selected argmax is the mask token, that call need not change the state. Thus the exact state-change count in part (b) additionally requires the proposition's non-mask assumption, while the output-invariance conclusion still holds. Equal accuracies in Table 14 therefore do not independently establish convergence. Remasking can erase commitments, and uniform-state sampling is time-conditioned and stochastic. Both execute T calls without this cap, although those calls need not all change the state or improve accuracy.

B TRAINING OBJECTIVES AND ALGORITHMS

This section collects the continuous and discrete objectives and the training and inference algorithms. Task serialization, model settings, and checkpoint selection follow in Appendix C.

B.1 CONTINUOUS CODEC AND OBJECTIVE

Each continuous codec is a β-VAE trained jointly on prompt and trace spans. The encoder first contextualizes the full token span with a non-causal transformer and then maps every fixed block of R tokens to one Gaussian latent. For an entangled code (K=1), the contextualized tokens in each block are mean-pooled before predicting the posterior mean and log variance. For a factorized code (K>1), each block is divided into K equal token sub-blocks that are mean-pooled; a shared projection maps each pooled sub-block to a d_c/K-dimensional posterior, and the K outputs are concatenated. The non-autoregressive decoder uses learned positional queries that self-attend and cross-attend to the latent sequence and predicts all output tokens in parallel. Factorization retains global contextualization unless the blockwise restriction is enabled. It structures the pooled representation without making the sub-slots informationally independent. The blockwise control restricts both encoder and decoder attention as specified in Appendix D.3.

For a span x, the codec loss is

$$\mathcal{L}_{\text{codec}}(x) = \text{CE}(\hat{x}, x) + \beta \ \text{KL}_{\text{FB}}(q_\phi(z \mid x) \,\|\, \mathcal{N}(0, I)) \,, \tag{1}$$

and the training loss is the sum of this objective over the prompt and trace spans. We use β=0.05 and a free-bits parameter of 0.5. In the implementation, this parameter is an elementwise 0.5-nat floor on the Gaussian KL before summing over latents and latent dimensions. Cross-entropy averages over all token positions, including padding.

B.2 Two-stage training and generation

Algorithms 1 and 2 summarize training and generation for the primary S_5 task, including the codec inclusion criterion. Training has two separate stages: codec training and validation, followed by generator training with the codec frozen. The task sampler constructs a token prompt p and fixed-length trace y, without natural-language instructions. The prompt encoder $\bar{E}_c$ equals $\bar{E}_\phi$ except in controls that use a separate prompt codec. Generator inputs use standardized latents. The vector b is AR's learned beginning-of-answer input, and a_j denotes the backbone output used to predict latent j. The codec chain-length policy P_{AE} varies by configuration (Appendix C.2). Generator MSE losses average over latent dimensions and target positions. All losses average over the minibatch. In the codec loss, j indexes latent positions and ℓ indexes dimensions, so the free-bits floor is applied to each scalar posterior coordinate.

Both samplers use fixed generation budgets. AR emits m latents, with J local Euler updates per emission. Continuous diffusion makes T full-sequence denoising calls. The prompt latents remain fixed throughout; AR appends generated latents to its prefix, while diffusion updates its trace latents and self-conditioning estimate.

Algorithm 1 Training the frozen latent space and matched continuous generators

Require: Task sampler $\mathcal{Q}$; chain-length policies $\mathcal{D}_{\text{train}}, P_{\text{AE}}$; codec settings R, K, β, f and attention scope; update budgets N_{AE}, N_G

Ensure: Frozen standardized encoder $\bar{E}_\phi$, decoder Dec_ψ, and a trained generator

Initialize codec parameters (ϕ, ψ) with posterior $q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \text{diag}\,\sigma^2_\phi(x))$.
$\mathcal{L}_{\text{codec}}(x) \leftarrow \text{CE}(\text{Dec}_\psi(z), x) + \beta \sum_{j,\ell} \max(\text{KL}(q_{\phi,j\ell}(z_{j\ell} \mid x) \| \mathcal{N}(0,1)), f)$, with $z \sim q_\phi(z \mid x)$
for $u = 1, \ldots, N_{\text{AE}}$ **do**
  Draw $D \sim P_{\text{AE}}$ and $(p, y) \sim \mathcal{Q}(\cdot \mid D)$.
  Update (ϕ, ψ) using $\mathcal{L}_{\text{codec}}(p) + \mathcal{L}_{\text{codec}}(y)$.
end for
From posterior means of both spans in eight fresh minibatches, compute per-dimension (μ_z, σ_z); clamp $\sigma_z \geq 10^{-4}$.
Define $\bar{E}_\phi(x) = (\mu_\phi(x) - \mu_z)/\sigma_z$; freeze E_ϕ, Dec_ψ, μ_z, and σ_z.
for $D \in \{2, 4, 8, 16, 32\}$ **do** ▷ codec inclusion criterion
  On 256 fresh instances, require $\text{mean}\,\textsc{Check}(p, \arg\max \text{Dec}_\psi(\mu_\phi(y))) \geq 0.95$; otherwise exclude the codec.
end for
for $u = 1, \ldots, N_G$ **do**
  Cycle D through $\mathcal{D}_{\text{train}}$, sample $(p, y) \sim \mathcal{Q}(\cdot \mid D)$, and set $c \leftarrow \bar{E}_c(p)$, $x_0 \leftarrow \bar{E}_\phi(y)$.
  if training latent AR **then**
    For every target position j, compute $a_j \leftarrow A_\theta([c, b, x_{0,<j}])_{\text{last}}$ by teacher forcing.
    Draw $\tau_j \sim \mathcal{U}[0,1]$, $\epsilon_j \sim \mathcal{N}(0, I)$, and set $u_j = (1 - \tau_j)x_{0,j} + \tau_j \epsilon_j$.
    Update (θ, ω) using $\text{mean}_j \,\text{MSE}(H_\omega(u_j, a_j, \tau_j), \epsilon_j - x_{0,j})$.
  else ▷ training continuous latent diffusion
    Draw $t \sim \mathcal{U}[0,1]$, $\epsilon \sim \mathcal{N}(0, I)$, and set $z_t = (1 - t)x_0 + t\epsilon$.
    With probability $1/2$, set $s \leftarrow \text{stopgrad}(F_\theta(z_t, t; c, 0))$; otherwise set $s \leftarrow 0$.
    Update θ using $\text{MSE}(F_\theta(z_t, t; c, s), x_0)$. ▷ x_0 prediction
  end if
end for
return $\bar{E}_\phi$, Dec_ψ, and the trained (A_θ, H_ω) or F_θ

Algorithm 2 Conditional latent generation and exact final-state evaluation

Require: Token prompt p; prompt encoder $\bar{E}_c$; trace decoder Dec_ψ and latent statistics (μ_z, σ_z); trace length L_y; compression ratio R; AR head steps J or diffusion steps T
Ensure: Generated token trace $\hat{y}$ and its graded final state $\hat{h}_D$

$c \leftarrow \bar{E}_c(p)$; $m \leftarrow \lceil L_y/R \rceil$ ▷ the condition c remains fixed throughout generation
if latent AR **then**
  $q \leftarrow [c, b]$
  for $j = 1, \ldots, m$ **do**
    $a_j \leftarrow A_\theta(q)_{\text{last}}$; draw $u^{(0)} \sim \mathcal{N}(0, I)$.
    for $r = 0, \ldots, J-1$ **do**
      $\tau_r \leftarrow 1 - r/J$; $u^{(r+1)} \leftarrow u^{(r)} - J^{-1} H_\omega(u^{(r)}, a_j, \tau_r)$.
    end for
    $\hat{z}_j \leftarrow u^{(J)}$; $q \leftarrow [q, \hat{z}_j]$. ▷ previous latents are never revised
  end for
else ▷ continuous latent diffusion
  Draw $z^{(0)} \sim \mathcal{N}(0, I)^{m \times d_c}$; set $s^{(0)} \leftarrow 0$.
  for $r = 0, \ldots, T-1$ **do**
    $t_r \leftarrow 1 - r/T$; $\hat{x}_0^{(r)} \leftarrow F_\theta(z^{(r)}, t_r; c, s^{(r)})$.
    $z^{(r+1)} \leftarrow z^{(r)} - T^{-1}(z^{(r)} - \hat{x}_0^{(r)})/t_r$; $s^{(r+1)} \leftarrow \hat{x}_0^{(r)}$.
  end for
  $(\hat{z}_1, \ldots, \hat{z}_m) \leftarrow z^{(T)}$. ▷ all positions are revised jointly at each step
end if
$\hat{y} \leftarrow \arg\max \mathrm{Dec}_\psi(\sigma_z \odot \hat{z} + \mu_z, L_y)$.
$D \leftarrow \#\{\text{non-pad operation slots in } p\}$; $\hat{h}_D \leftarrow \hat{y}[5(D-1)+1 : 5D]$.
return $\hat{y}$, $\hat{h}_D$, and $\textsc{Check}(p, \hat{y})$

B.3 Discrete representations, objectives, and samplers

Post-hoc codebook. The discrete models use a post-hoc k-means codebook to replace each sub-slot vector of the same frozen codec with a code token. The shared codebook has 2,048 entries of width $384/4 = 96$. Thus, the ten continuous trace latents become 40 discrete code tokens. Decoding replaces each token by its centroid, concatenates each group of four sub-slots, and applies the original frozen decoder. The codebook is fitted to standardized posterior-mean sub-slots from 128 batches of 128 programs, including both prompt and trace spans, using 60 Lloyd iterations. Chain lengths are sampled with probability proportional to D over 2–32. The same codebook encodes the prompt: its 34 tokens become $4\lceil 34/16 \rceil = 12$ code tokens.

Masked diffusion (MDLM/LLaDA NELBO). We compare the original masked cross-entropy baseline with a $1/t$-weighted MDLM/LLaDA NELBO estimator (Sahoo et al., 2024; Nie et al., 2025). For batch size B, target length L, mask indicator m_{ij}, mask probability t_i, and token loss ℓ_{ij} on the corrupted input, the implemented losses are

$$\mathcal{L}_{\mathrm{CE}} = \frac{\sum_{i,j} m_{ij} \ell_{ij}}{\sum_{i,j} m_{ij}}, \qquad \mathcal{L}_{\mathrm{NELBO}} = \frac{1}{B} \sum_i \frac{\sum_j m_{ij} \ell_{ij}}{L t_i}. \tag{2}$$

The CE baseline normalizes by the total realized mask count across the batch; the NELBO estimator weights each example by its noise level. These are different weighting schemes, so their expectations need not coincide. The CE implementation forces one mask in otherwise unmasked rows. The NELBO implementation instead leaves such rows at zero loss, samples noise levels with a low-discrepancy batch scheme, and clamps t_i below at 10^{-3} to control the reciprocal weight.

Uniform-state (editing) diffusion. The uniform-state models are time-conditioned. Under uniform replacement noise, even an observed token can be corrupted, so the denoising posterior depends on the noise level. We implemented the continuous-time NELBO from the reference UDLM implementation (Schiff et al., 2025). A float64 unit test compares it with the $\mathrm{d}t \to 0$ limit of the discrete-time posterior KL. The relative error is below 1% at $t \in \{0.15, 0.5, 0.9\}$ in this numerical check.

Sampling and training budgets. Absorbing sampling starts from a fully masked trace and commits the most confident predictions according to the cumulative quota in Proposition 2. Low-confidence remasking uses the same checkpoint but can mask previously committed positions again. Uniform-state sampling starts from uniformly random code tokens and resamples every position at each step using the time-conditioned reverse posterior. The prompt stays fixed under all three samplers.

Code-token generators use 16 layers of width 384, eight attention heads, batch size 128, and AdamW at 3×10^{-4}, with the same weight decay and gradient clipping as the continuous generators. Their base budget is 25k updates; the extended masked-NELBO and uniform-state controls use 75k. The earlier 9.5M token anchors use 8k or 24k updates, respectively. Appendix D.5 reports the results and separates the effects of objective, sampler, and budget.

C Experimental setup and model selection

C.1 Worked prompt and trace example

Prompt tokens. For the primary S_5 trace task, $D_{\max}$=32 and the raw prompt contains one beginning marker, exactly 32 operation slots, and one separator:

$$p(w_{1:D}) = [\langle \texttt{bos} \rangle, g_{w_1}, \ldots, g_{w_D}, \underbrace{\langle \texttt{pad} \rangle, \ldots, \langle \texttt{pad} \rangle}_{32-D \text{ slots}}, \langle \texttt{sep} \rangle]. \tag{3}$$

Here, each g_j denotes one operation token. There is no query token in trace mode. The corresponding permutations are $g_0 = (1, 0, 2, 3, 4)$, $g_1 = (1, 2, 3, 4, 0)$, $g_2 = (0, 2, 1, 3, 4)$, and $g_3 = (1, 0, 3, 2, 4)$, where a tuple lists the image of each input position. For the four-step program (g_0, g_1, g_2, g_3), the complete 34-token prompt is shown below. It contains 28 padding tokens; line breaks have no meaning to the model.

```
<bos> g0 g1 g2 g3 <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <sep>
```

These symbols map directly to a fixed task vocabulary before the encoder's embedding layer. We do not apply a natural-language subword tokenizer. The relevant vocabulary IDs are 0 for <pad>, 1 for <bos>, and 3 for <sep>. Operation tokens g0–g3 have IDs 5–8, and value tokens v0–v4 have IDs 9–13. The prompt therefore encodes as $[1, 5, 6, 7, 8, \underbrace{0, \ldots, 0}_{28 \text{ entries}}, 3]$.

Reference trace and grading. Starting from the identity, we apply each generator on the left: $h_i(j) = g_{w_i}(h_{i-1}(j))$. The four intermediate states, serialized as value tokens, are

$$\begin{aligned} h_1 &= [\texttt{v1}, \texttt{v0}, \texttt{v2}, \texttt{v3}, \texttt{v4}], \\ h_2 &= [\texttt{v2}, \texttt{v1}, \texttt{v3}, \texttt{v4}, \texttt{v0}], \\ h_3 &= [\texttt{v1}, \texttt{v2}, \texttt{v3}, \texttt{v4}, \texttt{v0}], \\ h_4 &= [\texttt{v0}, \texttt{v3}, \texttt{v2}, \texttt{v4}, \texttt{v1}]. \end{aligned}$$

Concatenating these states gives 20 value tokens. The training target then appends 140 <pad> tokens, representing the remaining 28 state slots, to obtain the fixed 160-token trace. The oracle grades the five tokens in h_4, at one-based target positions 16–20. It infers the chain length from the prompt rather than grading the padded end of the target. This reference trace is the supervised target, not a generator input at test time.

Inputs to the latent generators. With the default global codec at $R = 8$, the deterministic encoder produces $\lceil 34/8 \rceil = 5$ prompt latents $c_{1:5}$, and each generator produces $160/8 = 20$ trace latents. The encoder first contextualizes the 34 prompt tokens, then appends six zero vectors before block pooling. These vectors complete the last eight-position block; they are distinct from the 28 <pad> tokens already present in the raw prompt.

The first AR call receives $[c_{1:5}, b]$, where b is the learned beginning-of-answer vector; the next receives $[c_{1:5}, b, \hat{z}_1]$. Continuous diffusion keeps the same $c_{1:5}$ at every step and updates all 20 noisy trace latents jointly. Its self-conditioning input is initially zero and subsequently the previous clean estimate. Neither generator receives the reference trace at test time or constructs a new textual prompt after each call.

C.2 Codec training and validation

All codecs use total latent width d_c=384, eight attention heads, batch size 128, and AdamW with learning rate 3×10^{-4}, weight decay 0.01, and gradient clipping at 1.0.

After codec training, we compute a per-dimension latent mean and standard deviation from posterior means on eight newly sampled minibatches, using both prompt and trace spans, clamp standard deviations below at 10^{-4}, and freeze these statistics together with the encoder and decoder. All downstream generators use the deterministic posterior mean rather than sampling from the posterior.

Because task instances are generated procedurally online, we use an independent random-number stream rather than a fixed validation corpus. The automated reconstruction gate draws 256 fresh examples at each chain length $D \in \{2, 4, 8, 16, 32\}$ and evaluates the deterministic posterior-mean encode–decode path. We record prompt-token accuracy, trace-token accuracy, exact-trace accuracy, and task-level final-state accuracy. The gate uses the last metric and retains a codec only if it is at least 0.95 at every chain length. The reconstruction ceilings reported with the original generator grid repeat this evaluation on 512 fresh examples per chain length. The matched token–latent comparison instead uses its shared 2,048-program test set. We use "reconstruction ceiling" as shorthand for this measured gold-latent reference. It is not a mathematical upper bound on generator accuracy: a generator need not reproduce the encoder's latent for a trace to decode correctly. Final-state reconstruction also need not imply an exact full trace (Appendix E.2). Graph walking uses its task-specific chain lengths $D \in \{2, 4, 8, 12\}$ for the gate and the matched evaluations; earlier controls report $D \in \{2, 8, 12\}$.

Chain-length coverage. Late trace positions receive non-padding targets only from sufficiently long programs. Table 1 collects the retained training recipes. Length-biased sampling gives late trace positions greater coverage than cycling over the five generator-training lengths. AR and diffusion share the codec within each configuration; codec training is not identical across all compression ratios.

Table 1: Retained continuous-codec recipes. E/D denotes encoder/decoder layers. "Cycle" repeats $D = 2, 4, 8, 16, 32$; "biased" samples every integer length in the stated range with $P(D) \propto D$. Slashes list separate K values. All rows use latent width 384 and the optimizer settings above.

Task / attention	(R, K)	E/D	Updates	Chain lengths
S_5, global	$(1, 1), (4, 1)$	4/4	15k	Cycle
S_5, global	$(8, 1), (16, 1/2/4)$	6/6	45k	Biased, 2–32
S_5, global	$(32, 8)$	6/6	45k	Biased, 2–32
S_5, global	$(32, 2/4)$	6/12	55k	Biased, 2–32
S_5, blockwise	$(8, 1), (16, 1/4), (32, 8)$	6/6	45k	Biased, 2–32
Affine, global	$(16, 1), (32, 8)$	6/12	60k	Biased, 2–32
Walker, global	$(1, 1)$	6/6	30k	Biased, 2–12
Walker, global	$(3, 1), (6, 2)$	6/12	45k	Biased, 2–12

Decoder capacity at high compression. At R=32 the entangled codec does not pass the reconstruction gate (final-state reconstruction 0.000 at D=32), whereas the factorized codecs do: $K = 8$ reaches 1.000 with the six-layer, 45k-update recipe, while $K = 2$ and $K = 4$ reach 0.998 and 0.965 using a six-layer encoder, 12-layer decoder, and 55k updates. Reconstruction therefore depends on code structure and codec training as well as compression ratio. These $R = 32$ factorization controls hold generator training fixed but use different codec recipes.

The R=1 walker codec is used both as the lossless prompt control and as the uncompressed trace codec. The main walker compression comparison keeps the separate R=3 prompt codec fixed.

C.3 Generator training and model selection

The base latent generators use 16 transformer layers of width 384, eight attention heads, batch size 128, and 25k AdamW updates. The learning rate is 3×10^{-4}, weight decay is 0.01, and gradient clipping is 1.0. Training cycles over chain lengths $2, 4, 8, 16, 32$ for composition and $2, 4, 8, 12$ for graph walking. The 178M AR and 319M diffusion scale controls use width 768, 24 layers, a learning rate of 10^{-4}, and 2,000 warmup updates, with the same total update budget. We report the final training checkpoint, and no early stopping is used. The headline results hold one seed-0 codec fixed and vary only the generator seed over $0, 1, 2$, except that the main graph-walking diffusion results use the repeated cohort $3, 4, 5$ (Appendix D.4). For the headline three-seed summaries, the best diffusion denoising budget is selected separately for each generator checkpoint and chain length from the shared grid $T \in \{1, 16, 64, 256\}$. These best-T entries are maxima on the reported evaluation set, not performance at a budget selected on a separate validation set. Fixed-T curves report each budget directly; the main graph-walking comparison fixes T=256.

Table 2: **Primary S_5 latent-generation grid.** Parameter pairs denote latent AR / continuous latent diffusion sizes, excluding the frozen codec. AR totals include the flow head. Unless noted otherwise, evaluation uses 512 test programs per chain length.

Setting	Values	Setting	Values
Compression R	$1, 4, 8, 16, 32$	Base params. (AR / Diff.)	35.5M / 54.0M
States/latent ($R/5$)	$0.2, 0.8, 1.6, 3.2, 6.4$	Scaled params. (AR / Diff.)	178M / 319M
Sub-slots K (R=16/32)	$1, 2, 4$ / $2, 4, 8$	AR flow-head size	7.1M
Chain length D	2–32	AR flow-head steps J	4
Denoising steps T	$1, 16, 64, 256$	Test programs / chain length	512

C.4 Graph-walking task and serialization

The graph-walking task has five nodes per layer and $D_{\max}$=12. For each source node, a destination is sampled uniformly from the five nodes in the next layer. Thus, each layer defines a deterministic function, but the sampled function need not be a permutation. The prompt serializes the graph as a fixed-layout edge list. Each of the 12×5 source nodes has two (source, destination) slots; the walker fills only the first, and all other slots, including every slot beyond layer D, are padded. With a leading token and a four-token query that specifies the start node and D, the prompt has $1 + 240 + 4 = 245$ tokens. The trace records one node-ID token per transition and is padded to 12 tokens. Unlike the five-token permutation states, this format gives R states per latent: four latents at R=3 and two at R=6.

The main trace-compression comparison holds the R=3 prompt codec fixed at 82 prompt latents and varies only the trace codec between R=1 and R=3. Its training-seed cohorts and complete results appear in Appendix D.4; the prompt-switching and correct-prefix diagnostics appear in Appendix E.

D Additional results and controls

This section gives the numerical results and controls supporting RQ1–RQ5. Appendix C collects the experimental settings, Appendix B gives the objectives and algorithms, and Appendix E presents the diagnostic probes. Unless stated otherwise, evaluations use 512 programs per checkpoint and chain length. Values after $\pm$ are sample standard deviations across generator seeds, not confidence intervals. A value rounded to 1.00 need not indicate zero errors.

D.1 Token–latent comparison and grid replication

Full matched token–latent comparison. Table 3 lists all five cells of the matched protocol behind Figure 4: two token models (a 16-layer model with the latent backbone's width and depth, and the 20-layer model with the latent generator's parameter count) and latent AR at R=$1, 4, 8$. The backbone-matched 16-layer token model is slightly stronger than the 20-layer one at D=32 (0.990 versus 0.962), and uncompressed R=1 latents are less accurate than tokens (0.917 ± 0.072; one seed reaches 1.00 and two fall to 0.87–0.88), so the gain in Figure 4 supports the benefit of

moderate compression rather than a general advantage of continuous representations. Accuracies were recounted from the saved predictions, the gold traces were checked against an independent oracle, and every cell uses the same test programs. Token AR uses greedy decoding, while latent AR samples its flow head with J=4 steps. All models generate the complete padded trace. Codec parameters and pretraining are additional resources, so this comparison matches generator updates and model size, not total training compute or inference latency.

Table 3: Full matched token–latent AR comparison (three seeds, 2,048 test programs per chain length; final-state accuracy, mean ± SD). All cells reach 1.00 at $D \leq 4$.

Model	Params. (gen. + codec)	Emissions	D=8	D=16	D=32
Token AR, 16 layers	28.5M	160	1.000 ± 0.000	0.998 ± 0.003	0.990 ± 0.011
Token AR, 20 layers	35.6M	160	1.000 ± 0.000	0.995 ± 0.002	0.962 ± 0.022
Latent AR, R=1	35.5M + 17.0M	160	1.000 ± 0.001	0.998 ± 0.002	0.917 ± 0.072
Latent AR, R=4	35.5M + 17.0M	40	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
Latent AR, R=8	35.5M + 25.3M	20	1.000 ± 0.000	1.000 ± 0.000	0.994 ± 0.009

Headline latent-space grid. For the original latent-generator grid, we evaluate three generator seeds per configuration with one frozen codec. For diffusion, the best tested T is selected separately for each checkpoint and chain length. Table 4 reports the entangled-code (K=1) results at $D = 16, 32$ for the compression ratios that pass the reconstruction gate. Latent AR stays at or near the reconstruction ceiling at R=4, 8, carrying the 32-composition chain in 40 and 20 emissions, respectively. These are the 512-program evaluations used in Figure 6, not the 2,048-program reevaluation for the matched token–latent comparison in Table 3. The R=16 values retain their original sampling protocol; the intervention replication uses the fresh evaluation specified in Appendix D.3.

Table 4: Numerical results underlying Figure 6a. Entries report final-state accuracy averaged over three generator seeds with one frozen codec, and diffusion entries use the best reported T for each checkpoint. Each checkpoint is evaluated on 512 programs per chain length. Values after ± are sample standard deviations. Every cell has a measured reconstruction ceiling of 1.00.

Generator	$R = 1$	$R = 4$	$R = 8$	$R = 16$
Chain length $D = 16$				
Latent AR	0.997 ± 0.005	1.000 ± 0.000	1.000 ± 0.000	0.017 ± 0.009
Latent diffusion	0.992 ± 0.014	0.801 ± 0.025	0.514 ± 0.162	0.016 ± 0.006
Chain length $D = 32$				
Latent AR	0.915 ± 0.071	1.000 ± 0.000	0.999 ± 0.001	0.003 ± 0.003
Latent diffusion	0.721 ± 0.476	0.019 ± 0.009	0.010 ± 0.000	0.012 ± 0.003

Token-level trace anchors. These earlier token anchors use a 9.5M model and 8k updates, rather than the matched protocol in Figure 4. They are retained as historical controls and do not establish the main token–latent comparison. Token-level generators trained to produce the full trace vary substantially across seeds at their maximum tested chain length, D=16. Over three independently trained generators, token-AR final-state accuracy ranges from 0.20 to 1.00, whereas that of token masked diffusion ranges from 0.71 to 0.96 under the same evaluation.

Token-AR compounding calibration. For the low-accuracy token-AR run, a separate probe on 2,048 programs measures teacher-forced next-token accuracy of 0.982. Treating this aggregate rate as a constant probability of extending a correct prefix gives $0.982^{80} \approx 0.23$ over the 80-token trace (16 states × five tokens), close to the observed final-state accuracy of 0.20. This is a rough compounding calculation: teacher-forced accuracy is not a measured free-running conditional rate, and exact-prefix survival differs from final-state accuracy. The two stronger seeds have teacher-forced accuracy of at least 0.99999 and final-state accuracy near one.

No-trace token-AR control. This control predicts one queried entry of the final permutation as a single value token, without generating intermediate states. Accuracy falls from 1.00 at $D = 2$ to

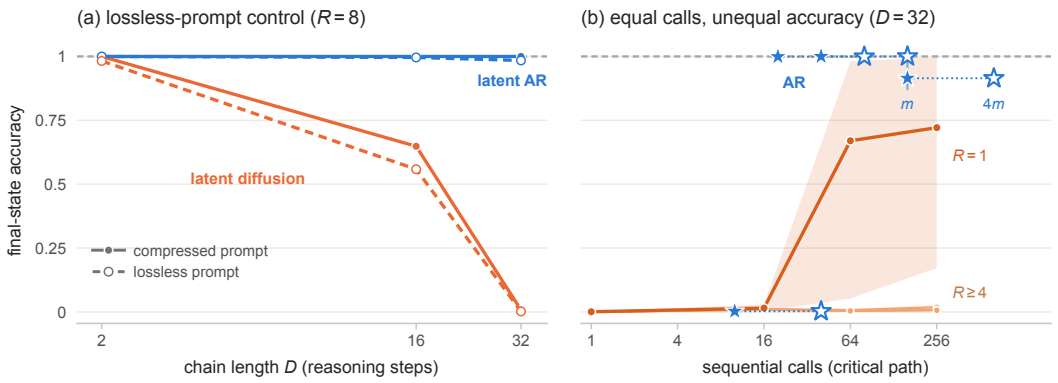

Figure 8: **Refinement controls for Section 3.4. (a)** At $R{=}8$ (one generator seed), replacing the prompt encoder with a lossless one (open markers) while keeping trace compression fixed does not remove the AR–diffusion gap. **(b)** At $D{=}32$, matching critical-path sequential calls does not align AR with diffusion under either AR cost convention: emissions m (filled stars) or $4m$ local flow-head updates (open stars). Panel (b) shows three-seed means and min–max ranges. Dashed lines mark the reconstruction ceiling.

0.39 at $D = 16$, compared with a chance rate of $1/5 = 0.20$. It illustrates the difficulty of longer composition chains for this short-answer model; it does not establish that every model requires an explicit trace.

D.2 Scale and Conditioning Controls

Diffusion scale control. In the single-checkpoint scale control, increasing the diffusion generator from 54M to 319M parameters raises accuracy at $R{=}8$, $D{=}16$ from about 0.65 to 0.89, but leaves $D{=}32$ accuracy near 0.01. The base model's three-seed mean is 0.51 (Table 4). On the entangled $R{=}16$ code, the scaled generator's original single-checkpoint prefix-survival diagnostic gives $p{=}0.763$ (Figure 9b). The complete three-seed scale control in Table 6 instead reports final-state accuracy: 0.016 ± 0.003 at $D{=}16$ and 0.011 ± 0.001 at $D{=}32$. Added capacity therefore does not recover accurate generation in this setting.

Easy-cell checks for the larger generators. The 178M AR generator over the $R = 8$ code reaches 1.00 at $D = 2, 16$ and 0.996 at $D = 32$, with a fitted prefix-survival factor of 0.99975. The 178M control at the composition-capacity limit ($R = 32$, $K = 8$) also scores 1.00 at $D = 2$. These checks show that the larger models learn the corresponding easier settings. They do not establish convergence at the harder compression ratios and chain lengths, where optimization remains a possible contributor to failure.

Parameter-comparable AR control. Increasing the $R = 1$ AR model to 24 layers (49.7M parameters, compared with diffusion's 54.0M) raises the fitted prefix-survival factor from 0.99358 to 0.99971. Measured full-prefix survival at $D = 32$ reaches 0.992. Thus, greater per-emission capacity can also improve chain survival without shortening the emitted sequence.

Conditioning-autoencoder (Cond-AE) control. The symmetric $R{=}8$ setting uses the same frozen codec for the prompt and the trace, so a long-chain failure could reflect information lost from the condition rather than difficulty generating the compressed trace. The control replaces only the prompt-side codec with a task-lossless $R{=}1$ codec and keeps the trace-side codec at $R{=}8$. Both frozen codecs share the same latent dimension, so the generator interface, architecture, and training protocol are unchanged.

This control uses one generator seed. Under it, continuous diffusion reaches 0.56 at $D{=}16$ and 0.00 at $D{=}32$, and does not improve over the same seed's symmetric run shown in Figure 8a, while the matched AR model reaches 0.98 at $D{=}32$. Removing prompt compression does not rescue the

long-chain diffusion cells. This argues against loss of task information in the prompt codec as the main explanation, while leaving the generator's use of that information unresolved.

Architecture control (plain blocks, in-context time token). The continuous diffusion generator uses DiT blocks with adaLN-Zero modulation to inject the noise level. We test whether the long-chain failure under compression persists with a different conditioning architecture. The control replaces these blocks with the AR generator's plain pre-norm blocks, using bidirectional attention, and supplies the noise level as one additional in-context token, following Peebles & Xie (2023). The objective, self-conditioning input, cross-attention to the prompt, and sampler remain unchanged. The resulting generator has approximately 39.5–39.6M parameters, compared with 54.0M for DiT and 35.5M for AR. Training follows the shared protocol: 16 layers, 25k updates, and three generator seeds. Table 5 reports final-state accuracy, selecting the best tested T per diffusion seed.

At D=32, the plain-block variant reaches 0.94 ± 0.05 at R=1 but only 0.02 ± 0.01 at R=4 and 0.01 ± 0.00 at R=8. The latter two values are close to the DiT results, while AR remains highly accurate at both compression ratios. At D=2, the variant's three-seed mean is 0.996 for R=1 and 1.00 for R=4, 8. At D=8, the corresponding means range from 0.905 to 0.997. Performance differs between the two configurations in other cells: at D=16, DiT performs better at R=4 (0.80 versus 0.12) and R=8 (0.51 versus 0.35), whereas at D=32, R=1, the plain-block variant has a higher observed mean and smaller seed spread (0.94 ± 0.05 versus 0.72 ± 0.48).

Thus, the long-chain failure under compression occurs with both tested conditioning architectures. These results do not isolate the effect of adaLN itself, since the parameter count also changes. Nor do they fully separate architecture from generation procedure: AR uses causal attention and reads the prompt as a prefix, whereas diffusion uses bidirectional attention and prompt cross-attention. The control establishes that replacing adaLN-Zero with time-token conditioning does not rescue the failure under the tested training and sampling budgets.

Table 5: Architecture control. Final-state accuracy of continuous diffusion with plain transformer blocks and an in-context time token (plain) against the DiT generator (adaLN) and the AR generator. Values are means and standard deviations over three generator seeds. For diffusion, the best tested T is selected separately for each seed. Reconstruction ceilings are 1.00 in every cell.

Generator	$R = 1$	$R = 4$	$R = 8$
Chain length $D = 16$			
Diffusion, plain	0.968 ± 0.055	0.122 ± 0.105	0.348 ± 0.060
Diffusion, adaLN	0.992 ± 0.014	0.801 ± 0.025	0.514 ± 0.162
Latent AR	0.997 ± 0.005	1.000 ± 0.000	1.000 ± 0.000
Chain length $D = 32$			
Diffusion, plain	0.938 ± 0.051	0.016 ± 0.005	0.009 ± 0.003
Diffusion, adaLN	0.721 ± 0.476	0.019 ± 0.009	0.010 ± 0.000
Latent AR	0.915 ± 0.071	1.000 ± 0.000	0.999 ± 0.001

Original refinement hypotheses. Before the initial comparison, we pre-registered three predictions from the intuition that compression removes AR emissions while diffusion can compensate with refinement:

***H1**: Increasing R hurts AR more than diffusion. **H2**: Increasing T recovers diffusion accuracy on longer chains. **H3**: Matching the number of sequential model calls aligns AR and diffusion accuracy.*

The results reverse H1, support H2 only over part of the tested range, and reject H3 under the two call-count conventions (Sections 3.1 and 3.4). We retain these original predictions here to distinguish them from the representation-focused questions developed through the subsequent controls.

D.3 Interventions at the two compression limits

Complete three-seed intervention evaluation. We evaluate all 22 configurations in Table 6 with generator seeds $0, 1, 2$ and one frozen codec per configuration. We train the 34 missing generators

with the original architecture, learning rate, warmup, FP32 precision, batch size 128, and 25,000 updates, and retain all final checkpoints regardless of outcome. The 32 existing checkpoints are reevaluated identically. Final-state accuracy uses 512 shared programs at each $D \in \{2, 16, 32\}$ (data seed 12345), sampling seed 20260925, and batch size 64. AR prefix-survival probes use 512 shared D=32 programs with data seed 4242. We fit p separately for each generator on points with $S(k) > 0.02$, then report the mean and sample standard deviation. Diffusion selects the best $T \in \{1, 16, 64, 256\}$ separately for each seed and chain length on the reported test set. All 66 checkpoints and 456 saved prediction files pass an independent recomputation of the metrics. These fresh stochastic evaluations can differ from the original grid in Table 4. Figure 1c and Figure 5a,b use these same fresh three-seed summaries. The additional runs, evaluations, and hardware checks consumed 29.19 V100-hours and 3.53 A100-hours in Slurm allocations; training alone accounted for 28.37 and 3.38 hours, respectively.

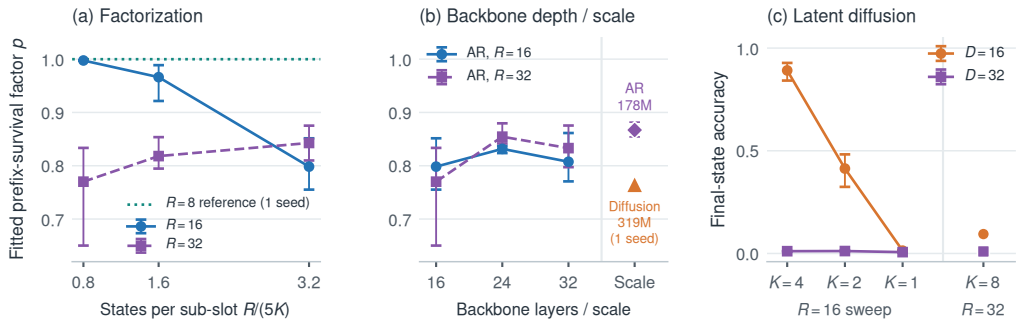

Figure 9: **Interventions at the two compression limits. (a)** Factorization moves the R=16 code-structure limit: at K=4 the R=16 code approaches the R=8 reference, whereas the R=32 sweep shows no consistent gain. A blockwise codec has the same effect (Appendix D.3). **(b)** Deeper AR backbones do not recover high prefix survival at either compression ratio. The plotted scale controls also remain below ceiling: AR at R=32 and diffusion at R=16. **(c)** Diffusion shows the same dose–response at D=16 but stays near zero at every factorization at D=32. In (a), the x-axis gives nominal states per sub-slot, $R/(5K)$, placing the R=16 sweep (K=1, 2, 4) and R=32 sweep (K=2, 4, 8) at the same positions. In (c), the three connected points form the R=16 sweep; the separate column is the R=32, K=8 control. In (b), the backbone-depth sweeps are drawn as lines and the single-configuration scale controls occupy a separate column. The survival-factor axis in (a) and (b) starts at 0.62 to retain all seed ranges. Points and error bars show three-seed means and min–max ranges under Appendix D.3, except the explicitly marked original single-checkpoint R=8 reference and 319M diffusion prefix-survival diagnostic.

Factorization dose–response. For the R=16 code, the fitted AR prefix-survival factor rises monotonically with K (Table 6; Figure 9a). Diffusion shows a parallel gain at D=16, where the K=4 code surpasses even the entangled R=8 code, but it remains flat at D=32 (Figure 9c). Factorization thus removes the code-structure limit for AR and substantially improves diffusion at moderate chain length, while the long-chain continuous-diffusion failure persists as a separate effect. At $R = 16$, $K = 4$, the original AR checkpoint scores 0.94 at $D = 32$ (Table 12); the fresh three-seed evaluation averages 0.930 with sample SD 0.008, as reported in Table 7. At K=2, the mean is 0.622, but the individual values 0.236, 0.797, and 0.834 reveal substantial seed sensitivity.

Backbone depth at both limits. The depth-only control keeps backbone width fixed at 384 and increases the number of layers from 16 to 24 and 32. The separate 5× parameter control increases both width to 768 and depth to 24 layers. Increasing depth alone overcomes neither limit. Specifically, the mean fitted prefix-survival factors for the entangled R=16 code are 0.798, 0.832, and 0.807 at 16, 24, and 32 layers. The validated R=32, K=8 codec similarly yields 0.770, 0.854, and 0.833 (Table 6; Figure 9b). The $R = 32$, $K = 8$ controls retain 1.00 accuracy at $D = 2$ across these backbone depths.

Table 6: Interventions at the two compression limits. All entries are means ± sample standard deviations over three generator seeds (0, 1, 2), with one frozen codec per setting. Block A reports the AR prefix-survival factor p at D=32. Block B reports diffusion final-state accuracy, selecting the best tested $T \in \{1, 16, 64, 256\}$ separately for each seed and chain length on the reported evaluation set. Evaluation follows Appendix D.3. L and W denote backbone layers and width.

Block A: latent AR

Setting	K	L	W	p at D=32
R=16				
Global, entangled	1	16	384	0.798 ± 0.049
Global, factorized	2	16	384	0.966 ± 0.039
Global, factorized	4	16	384	0.997 ± 0.000
Blockwise	1	16	384	0.998 ± 0.001
Deeper backbone	1	24	384	0.832 ± 0.007
Deeper backbone	1	32	384	0.807 ± 0.048
Larger generator	1	24	768	0.803 ± 0.047
R=32				
Global, factorized	2	16	384	0.843 ± 0.033
Global, factorized	4	16	384	0.818 ± 0.031
Global, factorized	8	16	384	0.770 ± 0.104
Deeper backbone	8	24	384	0.854 ± 0.022
Deeper backbone	8	32	384	0.833 ± 0.039
Larger generator	8	24	768	0.867 ± 0.014
Blockwise	8	16	384	0.856 ± 0.024

Block B: continuous latent diffusion

Setting	K	L	W	D=16	D=32
R=16					
Global, entangled	1	16	384	0.014 ± 0.002	0.007 ± 0.003
Global, factorized	2	16	384	0.413 ± 0.081	0.012 ± 0.003
Global, factorized	4	16	384	0.891 ± 0.044	0.011 ± 0.001
Larger generator	1	24	768	0.016 ± 0.003	0.011 ± 0.001
Blockwise	1	16	384	0.204 ± 0.065	0.008 ± 0.003
Blockwise	4	16	384	0.893 ± 0.042	0.021 ± 0.011
R=32					
Blockwise	8	16	384	0.946 ± 0.018	0.027 ± 0.011
Global, factorized	8	16	384	0.094 ± 0.013	0.010 ± 0.002

Table 7: **Factorization moves the R=16 compression limit, whereas a larger entangled-code generator does not.** Global-context codecs, with means ± sample SD over three generator seeds under Appendix D.3. The K=2 setting has substantial seed variability; every final checkpoint is included.

R=16 setting	Survival p D=32	Final-state D=16	Final-state D=32
Factorization sweep (35M AR)			
Entangled, K=1	0.798 ± 0.049	0.024 ± 0.017	0.003 ± 0.004
Factorized, K=2	0.966 ± 0.039	0.744 ± 0.433	0.622 ± 0.335
Factorized, K=4	0.997 ± 0.000	1.000 ± 0.000	0.930 ± 0.008
Larger entangled-code control (K=1)			
AR, 178M (5×)	0.803 ± 0.047	0.049 ± 0.004	0.006 ± 0.002

Seed robustness at the composition-capacity limit. At R=32 and K=8, three independently trained generators yield fitted prefix-survival factors of 0.770 ± 0.104 (mean and sample standard deviation across seeds in the fresh evaluation). Although the fitted factors vary, all three runs show the same qualitative failure at the composition-capacity limit: final-state accuracy is 1.00 at D=2, but ranges from 0.008 to 0.025 at D=16 and from 0.000 to 0.006 at D=32. Across the tested R=32 interventions, the highest mean p is 0.867 ± 0.014 for the 178M generator; its D=32 final-state accuracy remains 0.004 ± 0.002.

Granularity above the composition-capacity limit. Although all three R=32 codecs pass the reconstruction gate, the fitted prefix-survival factors show no consistent improvement with factorization, in contrast to the monotone rise obtained below the composition-capacity limit (Table 6; Figure 9a). The R=16, K=4 and R=32, K=8 settings have the same four-token sub-block size, but different sub-slot widths: 96 and 48 dimensions, respectively. Holding total latent width fixed therefore changes both the number of states represented per emission and the width available to each sub-slot. The comparison does not isolate the number of states per latent alone.

Blockwise codec. The headline codec is global on both sides (Appendix E.1). A blockwise codec gives each latent a strict information boundary: the encoder's self-attention is block-diagonal over R-token blocks, and the decoder's positional queries cross-attend only to the latent of their own block and self-attend only within it, so latent k is a function of its R tokens alone and block k is decoded from latent k alone. The codec recipe, generator architecture, and training budget are otherwise those of the headline cells. Blockwise codecs at R=8, R=16 (K=1, 4), and R=32, K=8 pass the reconstruction gate with a ceiling of 1.00 at every chain length, and the cross-talk probe finds no error outside the replaced block. A blockwise entangled R=32 codec reconstructs 0.98 at D=32 but fails the gate at D=2 (0.94), so no generator was trained on it. Table 8 compares the rest with the global codecs. At R=8, AR stays near the ceiling. At R=16, the blockwise *entangled* code recovers AR accuracy, reaching 0.953 ± 0.038 at D=32 over three generator seeds, with a fitted survival factor of 0.998 ± 0.001, matching the factorized global code: both restricting cross-block context and factorizing the latent can recover accurate generation at this compression ratio. At R=32, K=8, AR remains at 0.021 ± 0.010 at D=32 (p=0.856 ± 0.024), so the composition-capacity limit persists, although the blockwise codec raises intermediate-length accuracy for both paradigms (0.551 ± 0.072 for AR and 0.946 ± 0.018 for diffusion at D=16). Continuous diffusion stays near zero at D=32 under every blockwise codec.

D.4 Task-family contrasts

Affine contrast details. This contrast uses one latent-AR checkpoint per configuration. Under the entangled R=16 setting, the short-chain failure disappears: the affine model scores 1.00 at D=2 where S_5 reaches 0.67, although at greater chain lengths its prefix-survival factor is 0.82, still below ceiling. Under the R=32 and K=8 configuration, the affine model yields a fitted prefix-survival factor of 0.74, which falls directly inside the S_5 range across seeds. Its final-state accuracy at D=32 is 0.090, near the $1/10$ chance level for the ten affine permutations. The affine R=32, K=8 model

Table 8: Blockwise versus global codecs. Final-state accuracy at D=16, 32 (diffusion at its best tested T) and AR prefix-survival factor p at D=32. Unmarked entries are fresh three-seed means ± sample SD from Appendix D.3. Entries marked † retain their original evaluation: one checkpoint for R=8, and three seeds for blockwise AR at R=16, K=4, which is outside this replication. The single-checkpoint R=8 diffusion reference differs from the mean in Table 4.

		AR			diffusion	
Code	Codec	D=16	D=32	p	D=16	D=32
R=8, K=1	global	$1.00^{\dagger}$	$1.00^{\dagger}$	$1.000^{\dagger}$	$0.65^{\dagger}$	$0.01^{\dagger}$
	blockwise	$1.00^{\dagger}$	$1.00^{\dagger}$	$1.000^{\dagger}$	$0.37^{\dagger}$	$0.01^{\dagger}$
R=16, K=1	global	0.024 ± 0.017	0.003 ± 0.004	0.798 ± 0.049	0.014 ± 0.002	0.007 ± 0.003
	blockwise	0.992 ± 0.008	0.953 ± 0.038	0.998 ± 0.001	0.204 ± 0.065	0.008 ± 0.003
R=16, K=4	global	1.000 ± 0.000	0.930 ± 0.008	0.997 ± 0.000	0.891 ± 0.044	0.011 ± 0.001
	blockwise	$(0.98 \pm 0.02)^{\dagger}$	$(0.91 \pm 0.07)^{\dagger}$	$0.996^{\dagger}$	0.893 ± 0.042	0.021 ± 0.011
R=32, K=8	global	0.015 ± 0.009	0.003 ± 0.003	0.770 ± 0.104	0.094 ± 0.013	0.010 ± 0.002
	blockwise	0.551 ± 0.072	0.021 ± 0.010	0.856 ± 0.024	0.946 ± 0.018	0.027 ± 0.011

also scores 0.76 at D=2, with errors already in the first emitted state. The task families therefore share a long-chain failure without having identical short-chain behavior.

Walker details. The graph layout and serialization are specified in Appendix C.4. At $D = 12$, with compression ratio R=3 and a compressed prompt codec, latent diffusion leads free-running latent AR from a single sampling step onward and remains strong at $T = 64$ and $T = 256$, while AR stays at the $1/5$ chance rate (Figure 10a). Increasing the number of states per latent from three to six lowers the range of fitted prefix-survival factors across seeds from 0.50–0.91 to 0.35–0.48, and errors appear within the first emission.

A separate, single-seed lossless-prompt control further clarifies these dynamics. With compression ratio $R = 3$ and a lossless prompt codec, the teacher-forced probe scores between 0.998 and 1.00 across states 1–6. However, its performance drops sharply at the third emission. Both the long-chain collapse and the paradigm inversion persist under this control. Diffusion reaches 0.53 accuracy, compared with AR's 0.19, at D=12.

Compression control (walker). To separate the cost of packing several retrievals into one latent from the task itself, we hold the prompt codec fixed (the R=3 walker codec, 82 prompt latents) and vary only the trace representation. Latent AR and latent diffusion are trained at R=1, one state per latent and twelve latents at D=12, with the generator architecture, 25k updates, batch size, training chain lengths, and the corresponding within-paradigm seed cohort of the R=3 models. The R=1 trace codec is the lossless walker codec of the prompt-codec control above; it reconstructs the final state exactly at every evaluated chain length. We also train token-AR models on raw token prompts with the same task and data stream, following the protocol of Appendix D.1: a 16-layer model that matches the latent backbone (28.6M parameters) and a 20-layer model that matches the generator parameter count (35.7M). All systems are evaluated on the same 2,048 programs per chain length (Table 9). Token AR and R=1 latent AR reach 1.00 at every chain length across three seeds, with teacher-forced token accuracy and prefix survival of 1.00 at every position. At R=3, latent AR falls to chance at D=12 and is unreliable at D=4 and D=8, where individual seeds range from 0.22 to 1.00. At fixed T=256, diffusion improves from 0.403 to 0.597 at D=12, with gains in all three new seeds (Table 10). It also leads R=3 AR at T=1 (0.496 versus 0.201). At D=8, the mean increases from 0.604 to 0.643, but this smaller mean gain is not consistent across the three seeds (Table 11). Compression does not yield a uniform diffusion benefit: at fixed T=256, the new-cohort means decrease from 0.989 to 0.977 at D=2 and from 0.965 to 0.907 at D=4. At D=12, trace compression reverses the latent AR–diffusion ordering with the task, prompt encoder, and test programs held fixed, although diffusion remains well below perfect accuracy.

The R=1 codec uses a six-layer decoder and 30k updates rather than the twelve-layer, 45k recipe of the R=3 codec, and the 20-layer token model was trained with two accumulated micro-batches of 64 per 128-example update on 16 GB GPUs, which is equivalent up to floating-point summation. The

trace-codec differences mean this comparison does not isolate the number of retrievals per emission from every aspect of the representation.

Table 9: Walker compression control: final-state accuracy on the same 2,048 programs per chain length, three-seed means with the per-seed minimum–maximum in parentheses where seeds differ after rounding. Latent models share the R=3 prompt codec. Token AR uses raw token prompts. Diffusion uses seeds $3, 4, 5$ and the best $T \in \{1, 16, 64, 256\}$ per seed; AR uses seeds $0, 1, 2$. In contrast, Figure 7a fixes T=256.

System	D=2	D=4	D=8	D=12
Token AR, 16 layers	1.00	1.00	1.00	1.00
Token AR, 20 layers	1.00	1.00	1.00	1.00
Latent AR, R=1	1.00	1.00	1.00	1.00
Latent AR, R=3	0.94 (0.83–1.00)	0.78 (0.35–1.00)	0.53 (0.22–0.99)	0.20 (0.20–0.21)
Latent diffusion, R=1	0.99 (0.97–1.00)	0.96 (0.93–0.99)	0.61 (0.50–0.81)	0.41 (0.35–0.45)
Latent diffusion, R=3	0.98 (0.93–1.00)	0.91 (0.82–0.99)	0.64 (0.50–0.91)	0.60 (0.52–0.72)

Table 10: Fixed-step latent diffusion on graph walking. Means and min–max ranges over seeds $3, 4, 5$ use the same 2,048 programs per chain length as Table 9. The final column supplies the diffusion points in Figure 7a.

D	R	T=1	T=16	T=64	T=256
2	1	0.66 (0.42–0.99)	0.97 (0.92–1.00)	0.99 (0.96–1.00)	0.99 (0.97–1.00)
2	3	0.91 (0.83–0.97)	0.97 (0.91–1.00)	0.97 (0.93–1.00)	0.98 (0.93–1.00)
4	1	0.38 (0.35–0.44)	0.83 (0.66–0.98)	0.95 (0.89–0.98)	0.96 (0.93–0.99)
4	3	0.74 (0.57–0.93)	0.86 (0.76–0.99)	0.90 (0.81–0.99)	0.91 (0.82–0.99)
8	1	0.40 (0.36–0.43)	0.52 (0.48–0.58)	0.58 (0.49–0.74)	0.60 (0.50–0.81)
8	3	0.44 (0.39–0.48)	0.61 (0.48–0.85)	0.64 (0.50–0.89)	0.64 (0.50–0.91)
12	1	0.35 (0.33–0.37)	0.39 (0.33–0.42)	0.40 (0.34–0.44)	0.40 (0.35–0.45)
12	3	0.50 (0.47–0.52)	0.57 (0.52–0.66)	0.59 (0.52–0.70)	0.60 (0.51–0.72)

Reported training seeds (walker). We repeated the unchanged diffusion training recipe with seeds $3, 4, 5$ after inspecting an earlier cohort. The graph-walking diffusion figures and tables consistently report this complete repeat cohort, including its denoising-step sweep and context interventions. The AR results continue to use seeds $0, 1, 2$; matching concerns the training protocol and test programs, not the numerical seed IDs across paradigms. Within diffusion, both trace compression ratios use the same three training seeds.

Table 11 gives each reported seed at fixed T=256. Compression improves accuracy at D=12 in all three runs. At D=8, seed 3 decreases while seeds 4 and 5 increase, so the small mean gain does not establish a consistent benefit across training seeds. The earlier cohort gave a different mean D=8 compression effect and is retained in the run artifacts to be released; it is not pooled into the reported three-seed summaries.

Table 11: Per-seed diffusion results for the reported repeat cohort at fixed T=256. All checkpoints use the same 2,048 programs per chain length. The final row gives the mean ± sample standard deviation.

	D=8		D=12	
Training seed	R=1	R=3	R=1	R=3
3	0.812	0.501	0.453	0.515
4	0.506	0.910	0.348	0.719
5	0.496	0.519	0.407	0.559
Mean ± SD	0.604 ± 0.179	0.643 ± 0.231	0.403 ± 0.052	0.597 ± 0.108

Here, prompt compression can be varied independently of trace compression. Its effect depends on the trace representation. For the R=3 trace model, the lossless context removes the early-state errors of the seed-0 model. The three-seed probe shows that these early errors are not universal (Table 17). For the R=6, K=2 trace model, the lossless context hurts. Encoding the edge list as 245 uncompressed latents instead of 41 compressed ones lowers accuracy at D=2 from at least 0.93 to 0.15. The fitted prefix-survival factor rises as prompt compression increases (Figure 10b). In this tested R=6 configuration, fewer, more compressed context latents support better performance.

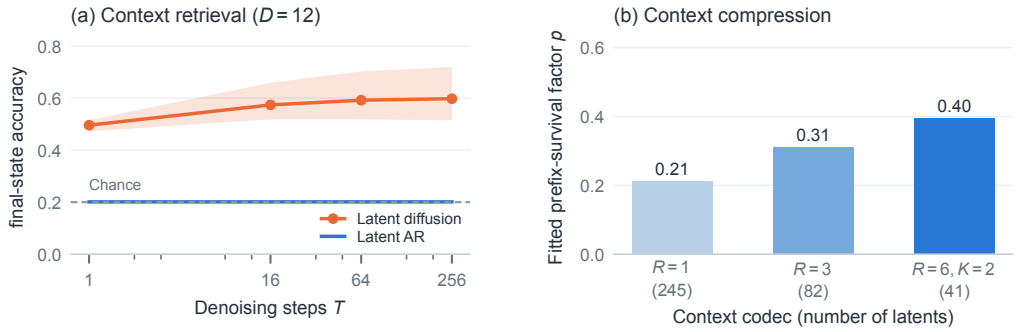

Figure 10: **Walker task: paradigm inversion and context compression.** **(a)** At D=12, with compression ratio R=3 and a compressed prompt codec, latent diffusion leads free-running latent AR from a single sampling step onward and performs similarly at $T = 64$ and $T = 256$; AR sits at the $1/5$ chance line. Lines are three-seed means and bands span the seed range on 2,048 shared programs. Diffusion uses seeds $3, 4, 5$ and AR uses seeds $0, 1, 2$. **(b)** For the R=6, K=2 trace model the fitted prefix-survival factor rises as prompt compression increases (one generator per configuration).

D.5 Discrete objectives and sampler results

These experiments support RQ4 by testing whether the continuous-diffusion failure persists with discrete representations and different samplers. They complement the representation-structure controls, but do not form a comparison in which only discretization changes.

Discrete code tokens avoid much of the continuous-diffusion collapse. At R=16, K=4, and D=32, discrete masked diffusion reaches 0.64 final-state accuracy, compared with 0.01 for continuous diffusion and 0.68 for code-token AR (Figure 11a). The code-token construction and training objectives are specified in Appendix B.3. The comparison also changes the objective, sampler, training budget, and sequence granularity (each sub-slot becomes a generated position), so it does not isolate discretization as the cause of the recovery.

Neither remasking uncertain tokens nor a uniform-state kernel that keeps all tokens editable (Schiff et al., 2025; Sahoo et al., 2025) improves on the best absorbing masked-diffusion accuracy in our controls (Figure 11b). For the absorbing sampler, extra steps leave the output unchanged once the schedule reveals at most one token per step (Proposition 2). The following controls detail the objectives, training budgets, and full step sweeps.

Masked-diffusion objective and training budget. At the base training budget, token-anchor NELBO has a three-seed mean of 0.14 at $D = 16$, compared with 0.85 for masked CE. A single-checkpoint control at three times that budget reaches 1.00 (Table 13). We use this extended budget for code-token masked diffusion, which reaches 0.64 at $D = 32$; the base-budget CE control scores 0.27. Because objective and budget both change in this comparison, the gain cannot be assigned to the loss alone. The masked-diffusion results in Tables 12 and 14 use the NELBO checkpoints, while the original token anchors use masked CE.

Uniform-state objective comparison. The NELBO model reaches the ceiling on the 14-token anchor alphabet. On the 2048-way code alphabet, at three times the base budget, it reaches 1.00 at

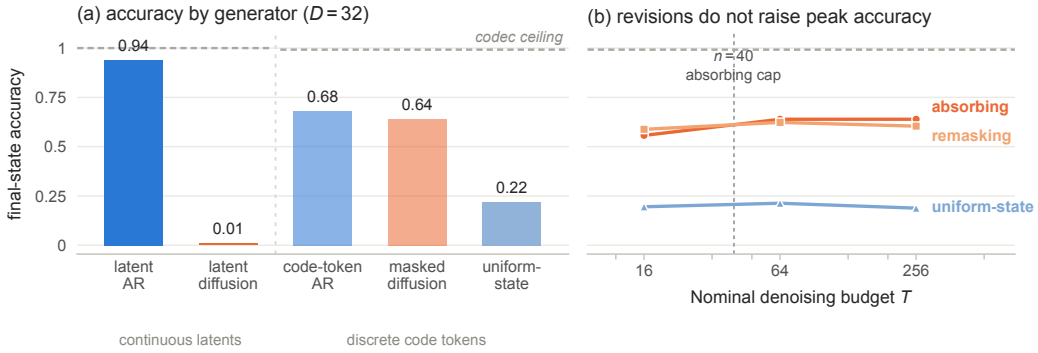

Figure 11: **Discrete code-token generators avoid much of the continuous diffusion failure.** R is tokens per continuous latent, K is sub-slots per latent, and D is chain length; the code-token models generate one token per sub-slot. Both panels use $R = 16$, $K = 4$, and $D = 32$. **(a)** Final-state accuracy for five generators derived from the same frozen factorized code, one checkpoint per generator. **(b)** Discrete sampler accuracy across denoising budgets. Discrete diffusion uses $3\times$ the base training budget of its model family (Appendix B.3).

Table 12: Five generators using representations derived from the same frozen factorized R=16 code. Masked diffusion uses the MDLM/LLaDA NELBO at the extended training budget. Uniform-state diffusion (Schiff et al., 2025) keeps every token editable at every step. All five reach 1.00 at D=2. Rows report individual checkpoints. The code-token rows use a k-means codec whose ceiling is 0.992 at D=32, compared with 1.000 for the continuous codec. The latent-AR row is the original checkpoint, not the three-seed fresh-evaluation mean of 0.93 in Table 7.

Generator	D=16	D=32
Latent AR (flow head)	1.00	0.94
Continuous latent diffusion (best T)	0.93	0.01
code-token AR	0.99	0.68
code-token masked diffusion (best T)	0.99	0.64
code-token uniform-state diffusion (best T)	0.99	0.22

$D = 2$ and 0.99 at $D = 16$ but only 0.02 at $D = 32$ (Table 13). These results show sensitivity to the training objective in the tested setting without identifying the optimization mechanism. The headline uniform-state results use UDLM's simple-CE training option at the same $3\times$ budget as code-token masked diffusion.

Uniform-state kernel cells. Both uniform-state objectives approach or reach the ceiling on the token anchor (Table 13). With discrete latent codes, the simple-CE model also reaches 0.99 at $D = 16$, but its $D = 32$ accuracy remains between 0.18 and 0.22 over the tested budgets from $T = 4$ to $T = 512$. At $T = 40$, it scores 0.22. The absorbing sampler reaches 0.64 at nominal $T = 64$ using only 40 nonzero-quota denoiser calls; Proposition 2 gives the same output at $T = 40$ on the same prompts. Low-confidence remasking peaks at 0.62 at $T = 64$, below absorbing unmasking, and becomes less accurate at $T = 256$ (Table 14). Allowing more revisions therefore does not close the observed accuracy gap in these controls.

D.6 AR FLOW-HEAD SAMPLING CONTROL

MAR-head inner-step control. The AR flow head uses J Euler steps to sample each latent. Its training loss does not depend on J, so the same checkpoints can be evaluated at $J \in \{1, 2, 4, 8, 16\}$ without retraining. Headline evaluation uses $J = 4$. The sweep covers headline checkpoints and

Table 13: Objective and kernel controls, as final-state accuracy. Block A is the 14-token anchor alphabet at its longest chain length, D=16. Block B is the 2048-way code alphabet at D=32. The base budgets are 8k updates for token anchors and 25k for code-token models. The 3× budgets are therefore 24k and 75k updates, respectively. Block A's two base-budget absorbing entries are three-seed means. All other entries use one checkpoint.

Block A: token anchor (14-token alphabet)

Objective (budget)	D=16
Absorbing kernel	
uniform masked CE (base budget)	0.85
LLaDA NELBO (base budget)	0.14
LLaDA NELBO (3× budget)	1.00
Uniform-state kernel	
simple CE (3× budget)	0.95
exact NELBO (3× budget)	1.00

Block B: 2048-way code alphabet

Objective (budget)	D=32
Absorbing kernel	
uniform masked CE (base budget)	0.27
LLaDA NELBO (3× budget)	0.64
Uniform-state kernel	
simple CE (3× budget)	0.22
exact NELBO (3× budget)	0.02

Table 14: Discrete sampler controls at R=16, K=4, and D=32 over the denoising-step budgets shared by all three evaluations. Entries report final-state accuracy over 512 evaluation examples from one checkpoint per trained model. Absorbing unmasking and remasking share a checkpoint.

Sampler	T=16	T=64	T=256
Absorbing unmasking	0.557	0.639	0.639
Low-confidence remasking	0.588	0.623	0.604
Uniform-state kernel	0.195	0.213	0.188

two factorized high-load configurations, with three seeds and 512 evaluation examples per seed (Table 15).

At $D = 32$, the $R = 4$ and $R = 8$ models remain near ceiling across the sweep. The $R = 16$, $K = 4$ model scores between 0.910 and 0.947, so its accuracy is not strictly invariant to the sampler. The entangled $R = 16$ model at $D = 2$ improves from 0.515 at $J = 1$ to 0.678 at $J = 4$, with a smaller further gain to 0.699 at $J = 16$. For the high-load $R = 32$, $K = 8$ model at $D = 8$, increasing J does not recover accurate generation. Thus, local head iterations affect some cells but do not remove the tested composition-capacity failure. We report them separately from the number of outer latent emissions.

The serial-call comparison in Figure 8b uses either the m outer emissions or the $4m$ local head updates at the default J=4. The latter is a head-update count, not the total number of module calls: each emission also requires one backbone pass. A diffusion step applies its 54.0M backbone to all positions; an AR emission uses a 28.4M backbone pass followed by sampling with its 7.1M head. These call counts therefore do not imply equal FLOPs or latency.

Table 15: AR flow-head sampling steps J: final-state accuracy, mean ± SD over three seeds and 512 programs per seed. Fresh head samples explain differences between the J=4 column and Tables 4 and 7.

Cell	J=1	J=2	J=4	J=8	J=16
R=4, D=32	0.996 ± 0.002	1.000 ± 0.000	0.999 ± 0.001	0.998 ± 0.003	0.998 ± 0.003
R=8, D=32	0.984 ± 0.002	0.993 ± 0.012	0.993 ± 0.012	0.993 ± 0.012	0.992 ± 0.014
R=16, K=4, D=32	0.930 ± 0.019	0.947 ± 0.005	0.933 ± 0.010	0.922 ± 0.007	0.910 ± 0.008
R=16 entangled, D=2	0.515 ± 0.028	0.628 ± 0.002	0.678 ± 0.011	0.690 ± 0.008	0.699 ± 0.011
R=32, K=8, D=8	0.447 ± 0.227	0.484 ± 0.264	0.462 ± 0.262	0.451 ± 0.253	0.439 ± 0.248

E MECHANISM PROBES

E.1 CODEC AND GRADING AUDITS

Noise tolerance. We inject Gaussian noise of scale σ into the deterministic standardized encodings of gold traces, decode the perturbed latents, and grade them with the task metric. At the largest tested scale, σ=0.8, final-state accuracy is at least 0.998 for $R \in \{1, 4, 8\}$ at $D \in \{2, 8, 32\}$. The tested affine codecs remain at or above 0.996 at $D \in \{2, 16, 32\}$. For the S_5 codecs at D=32, accuracy is 0.95 for the entangled R=16 code and 0.98 for R=32, K=8. These checks establish robustness to the tested perturbations around gold latents. They do not characterize decoder behavior on all errors a generator may produce. This probe uses a separate batch from the generator grid. Its unperturbed D=32 accuracies are 0.992 and 0.998 for those two codecs, respectively.

Decoder cross-talk probe. In the default codec, both encoder and decoder use global context. The encoder contextualizes the whole token sequence before pooling, and the decoder's positional queries cross-attend to every latent. A wrong early latent could therefore corrupt decoded states outside its own block, in addition to conditioning later AR emissions. To measure this channel without a generator, we encode gold traces at D=32, replace one latent with the gold latent at the same position from a different instance (a valid but wrong latent), keep every other latent gold, and decode. Table 16 reports state accuracy for blocks containing states 1, 8, and 16. Here, "inside" includes every state whose tokens overlap the replaced block. The other columns score the next two complete states, all later complete states, and the final state. Errors outside the replaced span are most pronounced in the neighboring states of the entangled R=16 code. Accuracy before the span remains near one. Final-state accuracy is unchanged for all but two substitutions at R=32, where it falls from 1.000 to 0.996 (one of 256 examples). Thus, the global decoder can spread a latent error outside its nominal span, although the measured effect on the final state is small in this probe. This does not rule out larger effects from other latent errors or from multiple incorrect latents during free-running generation. The blockwise control removes this channel by construction (Appendix D.3).

Table 16: Decoder cross-talk probe at D=32 over 256 gold traces. One latent is replaced by a valid but wrong latent and everything else stays gold. Entries are state accuracies after decoding, reported as the range over the three replaced blocks. Unmodified reference sequences have final-state accuracy 0.996 for the entangled R=16 code and 1.000 otherwise. Their mean state accuracy is at least 0.999.

Codec	inside block	next two states	all later states	final state
R=1	0.77–1.00	1.00	1.00	1.00
R=4	0.51–0.62	1.00	1.00	1.00
R=8	0.06–0.20	0.83–0.92	0.98–0.99	1.00
R=16, K=1	0.05–0.30	0.16–0.89	0.87–0.99	0.996
R=16, K=4	0.07–0.16	0.77–1.00	0.97–1.00	1.00
R=32, K=8	0.04–0.13	0.90–0.98	0.99–1.00	0.996–1.000

Grading audit (leakage canary). Pairing each generated trace of a trained headline model with a different instance's prompt collapses accuracy from 1.00 to 0.01, close to the chance rate of $1/120 \approx 0.008$. This confirms that a generated answer must agree with the supplied prompt to receive credit; the oracle recomputes the target from that prompt rather than treating the generated trace as gold.

E.2 CHAIN AND EMISSION-LOCALIZATION PROBES

Chain survival. For each free-running sample, the probe records whether every state through position k is correct. Averaging this indicator gives the empirical survival curve

$$S(k) = P(\hat{h}_1 = h_1, \ldots, \hat{h}_k = h_k).$$

We fit $\log S(k) \approx a + k \log p$ by ordinary least squares over positions with empirical survival above 0.02. The intercept a is free; p summarizes the slope of the surviving portion of the curve. If fewer

than two positions pass this threshold, the probe does not report a fitted factor. This statistic is neither an average state accuracy nor a directly measured probability of a correct emission. It also counts errors that the decoder propagates from a wrong latent into the following block, which the cross-talk probe finds mainly for the entangled R=16 code (Appendix E.1).

In the original single-checkpoint probes, the fitted factors are 0.99358 for the 35.5M R=1 model, 0.99992 for R=4, and 0.99985 for R=8. These probes use a separate evaluation batch from the matched token–latent comparison in Figure 4. Values for the parameter-comparable control and for the two compression limits are reported in Appendix D.2 and Table 6, respectively. For the R=1 model the unit-intercept approximation gives $0.99358^{32} \approx 0.81$, compared with the measured full-prefix survival $S(32) = 0.867$ on the same samples. It captures the scale of decay but is not the endpoint of the fitted curve or an independent prediction of final-state accuracy.

Reconstruction must also be checked at the level of the metric being interpreted. In the matched 2,048-program evaluation at D=32, the R=8 reference latents decode to the correct final state on every program, but the entire active trace is exact on only 0.973 of programs. The reconstruction ceiling for final-state accuracy therefore does not imply perfect prefix reconstruction. Some prefix errors can arise in the codec even without generation error.

Teacher-forced emission probe. In this probe, the causal prefix consists of gold latents. Each next latent is sampled from the head, inserted into the gold sequence, decoded, and graded on states fully contained within its nominal token span. As a calibration, S_5 at R=8 scores 1.00, matching its free-running accuracy. At R=32, K=8, the teacher-forced per-state accuracy, averaged over all fully contained states, is 0.42 for the checkpoint shown in Figure 5c. It is not directly comparable to its fitted prefix-survival factor p=0.825: the former averages state accuracies in the substitution probe, whereas the latter summarizes the log-slope of the free-running prefix survival curve above a threshold. Errors therefore persist even when the generator receives a correct prefix. Other latent positions remain gold during decoding, so this is a conditional diagnostic, not a full-generation accuracy estimate.

The D=32 permutation trace has 160 tokens. At R=32, its five latent spans each cover 32 tokens, whereas each state h_i occupies five tokens. Since these boundaries do not align, the spans contain $6, 5, 6, 5, 6$ complete states, respectively. The boundary-crossing states $h_7, h_{13}, h_{20}, h_{26}$ are excluded from this local diagnostic and shown as hatched cells in both adjacent rows of Figure 5c. The colored cells report the accuracy of each complete state, and the overall 0.42 averages all 28 scored states.

On graph walking, the R=1 trace model with the fixed R=3 prompt encoder scores 1.00 at every state in all three seeds (Figure 7b). We evaluate the R=3 probe for all three training seeds on the same 512 programs at D=12, using the same probe with data seed 31415 and sampling seed 20260924. The latter is reset for each checkpoint. Seed 0 already errs in the first latent, whereas seeds 1 and 2 are exact there (Table 17). Thus, early errors are not universal across seeds. Nevertheless, all three score only 0.21–0.22 on the last state of the final latent despite receiving ground-truth prefixes. Figure 7b shows the three-seed means and min–max ranges.

Table 17: Per-seed correct-prefix probe at R=3, D=12 on graph walking. Each triple gives the three state accuracies within one predicted latent. All checkpoints use the same 512 programs.

Seed	z_1	z_2	z_3	z_4
0	(0.79, 0.78, 0.19)	(1.00, 0.47, 0.31)	(1.00, 0.26, 0.15)	(1.00, 0.27, 0.22)
1	(1.00, 1.00, 1.00)	(1.00, 1.00, 0.98)	(1.00, 1.00, 0.81)	(1.00, 1.00, 0.21)
2	(1.00, 1.00, 1.00)	(1.00, 0.71, 0.33)	(0.95, 0.60, 0.16)	(0.58, 0.20, 0.21)

The single-seed prompt-codec control in Figure 12 uses separate sampling draws. In that evaluation, seed 0 with compressed context scores 0.77, 0.80, and 0.20 across its first emission. With a lossless prompt, the first two emissions are nearly exact, but the third and fourth fall to near chance despite their ground-truth prefixes. In this control, removing prompt compression eliminates early errors without eliminating the long-chain failure. These diagnostics locate errors within generated latents. They do not isolate trace compression as the sole cause or measure a fixed number of retrieval operations per forward pass.

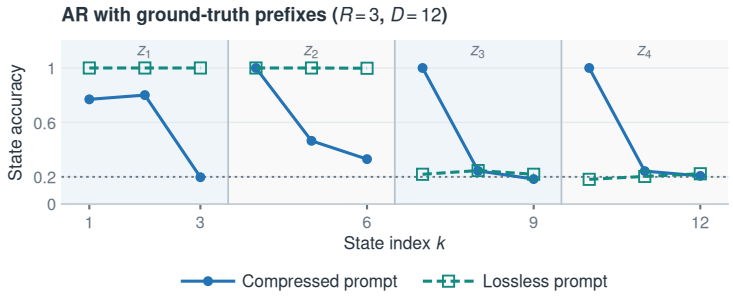

Figure 12: **Prompt compression changes where latent-AR errors occur.** Each latent is predicted from its ground-truth prefix and decoded in an otherwise ground-truth sequence. Shaded spans mark three-state latents at R=3, D=12. Each prompt setting uses one checkpoint. The lossless prompt removes early errors, but later latents remain inaccurate. The dotted line marks the 0.20 uniform-guess chance rate.

E.3 Repeated context use

Context-blinding intervention (walker). We use diffusion seeds $3, 4, 5$ and 512 shared programs at D=12 (data seed $12345 + D$, sampling seed 12345). Wrong context is the same-length batch rolled by one position, preserving chain length and the marginal prompt distribution. Conditions share problems and initial noise within each checkpoint. At T=256, switching from true to wrong context after any tested $k \leq 64$ leaves mean accuracy within 0.03 of the wrong-context floor. Switching from wrong to true instead recovers most of the unblinded accuracy (Table 18). This demonstrates continued prompt dependence and recovery from an incorrect start.

The first 64 steps of a 256-step trajectory span a different noise interval from a complete 64-step run, so this comparison does not isolate call count. Diffusion can revise all positions, whereas AR keeps emitted latents fixed. Repeated context use may contribute to denoising gains but does not explain the entire advantage, which is already present at T=1.

Table 18: Graph-walking context controls at D=12 (mean ± sample SD, seeds $3, 4, 5$; 512 programs). Left: true context throughout. Right: after k of 256 steps, switch true to wrong (late) or wrong to true (early). Conditions share problems and initial noise; sampling draws differ from Figure 10a.

Unblinded reference

T	Accuracy
1	0.499 ± 0.031
4	0.546 ± 0.065
16	0.562 ± 0.103
64	0.587 ± 0.118
256	0.590 ± 0.131

Switching step at T=256

k	Late blinding	Early blinding
1	0.185 ± 0.002	0.585 ± 0.135
2	0.184 ± 0.003	0.579 ± 0.140
4	0.187 ± 0.002	0.582 ± 0.136
8	0.181 ± 0.001	0.570 ± 0.124
16	0.184 ± 0.007	0.553 ± 0.121
64	0.204 ± 0.007	0.466 ± 0.037

Wrong context throughout: 0.191 ± 0.005; uniform-guess chance: 0.200.