REASON BEFORE YOU SPEAK WITH LATENT DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

The simple next-token prediction objective leads to the emergence of remarkable downstream language capabilities. However, recent work has uncovered surprising failure modes of this objective on simple algorithmic tasks. Autoregressive models struggle to plan ahead, maintain global coherence, and explore diverse solutions. These limitations stem from the myopic nature of next-token prediction. We demonstrate that latent diffusion can imbue autoregressive models with foresight. By jointly training next-token prediction with diffusion over a latent representation of the future, models can first develop a holistic plan before committing to discrete tokens. We evaluate on a suite of algorithmic tasks designed to stress-test planning, coherence, and creativity. Across all settings, latent diffusion substantially improves over autoregressive baselines. We validate our approach on a mathematical reasoning benchmark. Our results suggest that augmenting language models with latent planning capabilities offers a promising path toward improving coherence, creativity, and reasoning.

1 INTRODUCTION

The simple next-token prediction objective has proven remarkably effective: scaling it has produced complex capabilities in large language models (Brown et al., 2020). Yet the objective is inherently myopic: at each step, the model must commit to a token based only on the prefix it has observed, with no mechanism to consider how this choice constrains what follows. Because training conditions on ground-truth prefixes while inference conditions on the model's own, potentially incorrect outputs (Bengio et al., 2015), small mistakes can cascade and early commitments may prove irrecoverable.

These limitations manifest acutely in tasks requiring foresight, planning, and reasoning. On simple graph path-finding tasks, where the correct first token depends on the entire path structure, autoregressive models achieve near-perfect teacher-forced accuracy yet collapse to random guessing at inference time (Bachmann & Nagarajan, 2024).

The dominant approach to these limitations operates at post-training. Supervised fine-tuning teaches models to reason step by step through chains of thought (Wei et al., 2022; Abbe et al., 2024; Gandhi et al., 2025). Reinforcement learning further refines and extends these tendencies, producing "thinking models" (Guo et al., 2025). These interventions enable strong reasoning across mathematical, scientific, and coding domains, but the underlying architecture still generates tokens myopically and must simulate deliberation through expensive chain-of-thought generation.

Myopic generation also limits how models explore. An autoregressive model's samples can differ only in the tokens it samples, so diversity must be injected at the output layer, where it trades against quality: raising the temperature or widening the nucleus yields more varied but also more incoherent generations (Holtzman et al., 2020). Post-training resolves this tension in favor of quality. RLHF substantially reduces output diversity (Kirk et al., 2024), reinforcement learning with verifiable rewards improves pass@1 while base models overtake the trained models at large k (Chen et al., 2026), and independently trained models converge on similar responses to open-ended prompts (Jiang et al., 2026). Nagarajan et al. (2025) find that injecting noise at the input rather than the output can elicit diversity without hurting coherence, suggesting that the source of diversity matters.

We explore an alternative: augmenting the autoregressive objective with a mechanism for reasoning during *pre-training*. Continuous diffusion models offer a fundamentally different generative process:

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.

rather than committing to tokens sequentially, they iteratively refine a global view of the output from coarse to fine, planning the complete generation before filling in details. Recent work has extended latent diffusion to language generation (Lovelace et al., 2023; Zhang et al., 2023; Meshchaninov et al., 2025), and Lovelace et al. (2025) showed that a single architecture can first produce a continuous latent plan through iterative denoising and then use this plan to guide token generation. Because the plan is sampled, it also offers a different source of diversity: distinct plans yield distinct solutions, even when every token is decoded greedily. That work, however, extended a pre-trained model and compared it with other off-the-shelf pre-trained models on general language understanding, generation, and control. Whether latent planning confers a *capability* that autoregression lacks is a different claim, and demands a different standard of evidence: tasks on which next-token prediction demonstrably fails, and a comparison that holds data, parameters, and computation fixed. We provide the first such comparison, training both architectures from scratch under matched budgets, and ask: can latent diffusion actually improve the capacity for foresight, planning, and global coherence that next-token prediction struggles with, and can it improve the trade-off between quality and diversity?

We train autoregressive models with and without *latent diffusion planning* (LDP) from scratch under matched conditions, and evaluate on synthetic tasks that stress-test planning (star graph traversal), global constraint satisfaction (Sudoku), and creative exploration (sibling and triangle discovery), where the myopia of next-token prediction creates clear, measurable failures. Across all tasks, latent planning consistently improves over autoregressive baselines. On mathematical reasoning (GSM8K; Cobbe et al., 2021), our hybrid model outperforms an autoregressive baseline with nearly three times its active parameters at every sample budget, under both pass@k and majority voting. The difference lies in where diversity comes from. Decoding greedily from sampled plans, our model reaches higher coverage than any temperature and top-p setting of the autoregressive baselines while also achieving higher single-shot accuracy, and its programs remain valid at levels of diversity where those of standard autoregressive models break down. These results suggest that pre-training with latent planning is a promising path toward reasoning that complements autoregressive generation.

2 Background

Autoregressive Models. Autoregressive language models factorize a sequence $\mathbf{s} = (s_1, \ldots, s_n)$ as $P(\mathbf{s}) = \prod_{i=1}^{n} P(s_i \mid s_{<i})$ and are trained to maximize the likelihood of observed sequences, typically with Transformers (Vaswani et al., 2017; Brown et al., 2020). They generate one token at a time, each conditioned on all preceding tokens.

Diffusion Models. Diffusion models generate data by iteratively denoising Gaussian noise (Ho et al., 2020; Song et al., 2020). A forward process corrupts clean data $\mathbf{z}_0$ into increasingly noisy latents $\mathbf{z}_t = \alpha_t \mathbf{z}_0 + \sigma_t \boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for $t \in [0, 1]$, where $\alpha_t^2 + \sigma_t^2 = 1$ and the noise schedule, characterized by the log signal-to-noise ratio $\lambda_t = \log(\alpha_t^2/\sigma_t^2)$, ensures $\mathbf{z}_1 \approx \mathcal{N}(\mathbf{0}, \mathbf{I})$ (Kingma et al., 2021). A network trained to recover $\mathbf{z}_0$ from $\mathbf{z}_t$ reverses this process: generation starts from pure noise and denoises it step by step. For high-dimensional data, diffusion is often run in a learned latent space (Rombach et al., 2021). Unlike autoregressive models, diffusion models refine all dimensions of the output jointly, which enables global coordination but gives up conditioning on a partial output.

3 Latent Planning for Algorithmic Reasoning

We build on the multitask transformer of Lovelace et al. (2025), which learns autoregressive token generation and latent diffusion jointly during pre-training (Zhou et al., 2024). A continuous diffusion process over a latent representation of the output lets the model formulate a high-level plan before committing to individual tokens (Figure 1). We extend this architecture with role-specific *diffusion experts*, and refer to an autoregressive model trained this way as *AR w/ LDP*.

3.1 Model Architecture

Input Sequence. We structure the input sequence into three regions: a textual *prefix* containing the problem specification (e.g., a math word problem), a block of continuous *diffusion tokens* that serve as a latent planning buffer, and the textual *continuation* representing the target output (e.g., the

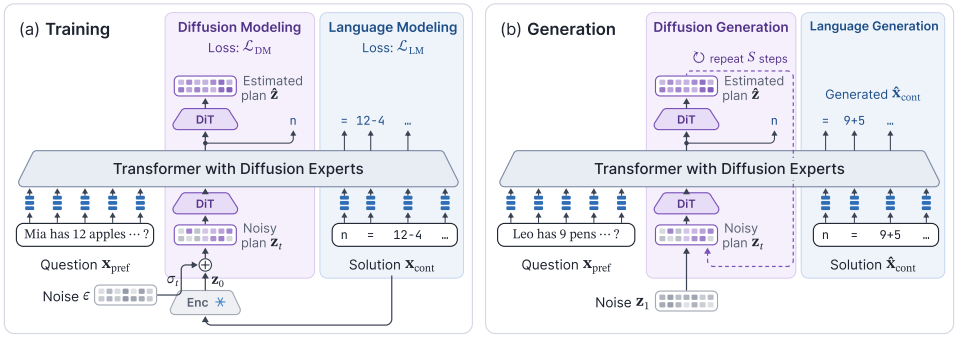


Figure 1: Training (left): the model jointly learns to predict next tokens and to denoise plan embeddings. Inference (right): a latent plan sampled via diffusion conditions autoregressive generation.

solution). Formally, for a sequence of length T:

$$\mathbf{x} = [\underbrace{x_1, \ldots, x_m}_{\mathbf{x}_{\text{pref}}} \mid \mathbf{z}_t \mid \underbrace{x_{m+1}, \ldots, x_T}_{\mathbf{x}_{\text{cont}}}] \tag{1}$$

where m is the prefix length and $\mathbf{z}_t \in \mathbb{R}^{k \times d_{\text{latent}}}$ is the noised latent plan, k latent vectors of dimension d_{latent} that lightweight input layers (below) map to the diffusion tokens.

Transformer with Diffusion Experts. Each transformer layer applies a *shared self-attention* layer over all positions, followed by one of two *expert* feed-forward networks (FFNs) chosen by each token's role (Figure 2): the *Language Expert* processes text tokens for next-token prediction, and the *Diffusion Expert* processes diffusion tokens for denoising. The two pathways exchange information only through the shared attention. As in mixture-of-experts architectures (Shazeer et al., 2017), this increases parameter capacity while holding FLOPs roughly constant, but it needs no learned routing because token types are known *a priori*. Because the diffusion tokens are denoised jointly rather than generated one at a time, we use a *block causal attention mask* (Figure 2, right). Diffusion tokens attend to one another bidirectionally regardless of position, enabling global reasoning about the latent plan, while text tokens keep standard causal attention.

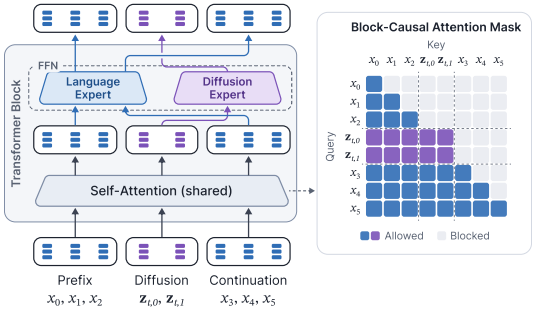


Figure 2: Transformer block with a diffusion expert FFN for diffusion tokens and a block causal attention mask that is bidirectional among them.

Latent Space. The diffusion process operates on a latent representation of the continuation, which should encode the full output to enable global planning while remaining compact for efficient diffusion (Rombach et al., 2021). We explore two instantiations:

Pooled Token Embeddings: For short, structured outputs, we embed the continuation with a dedicated low-dimensional embedding layer (separate from the LM embeddings) and mean-pool them into a single latent vector. For Sudoku, mean pooling collapses across puzzles, so we instead use strided pooling (kernel size 3), reducing 81 embeddings to 27 latent vectors. We L2-normalize the latent and rescale it to norm $\sqrt{d_{\text{latent}}}$, roughly that of a standard Gaussian sample.

Text Autoencoder Latents: For longer, more complex outputs (e.g., math word problems), mean pooling is too lossy. We instead train a text autoencoder following Meshchaninov et al. (2025) that compresses features of the continuation from a ModernBERT encoder (Warner et al., 2025), from variable length ($L \times d_{\text{enc}}$) into k latent vectors of dimension d_{latent} (Appendix E).

Diffusion Components. Lightweight Diffusion Transformer (DiT) layers (Peebles & Xie, 2023) on either side of the backbone map between the noised latent and the backbone's representations. They condition on the timestep t through adaptive normalization (Perez et al., 2018), so the network can adjust its behavior to the current noise level.

3.2 Training Procedure

Given a prefix $\mathbf{x}_{\text{pref}}$ and continuation $\mathbf{x}_{\text{cont}}$, we jointly optimize autoregressive and diffusion objectives (Figure 1, left). The language modeling loss is computed over the continuation, conditioned on the prefix and the noised latent, which the model sees through the diffusion tokens:

$$\mathcal{L}_{\text{LM}} = -\sum_{j=1}^{|\mathbf{x}_{\text{cont}}|} \log P_\theta(x_{\text{cont},j} \mid \mathbf{x}_{\text{pref}}, \mathbf{z}_t, \mathbf{x}_{\text{cont},<j}) \tag{2}$$

To train the model to denoise the latent plan, we follow standard practice: we sample a timestep $t \in [0, 1]$ and noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, form the noised latent $\mathbf{z}_t = \alpha_t \mathbf{z}_0 + \sigma_t \boldsymbol{\epsilon}$, and minimize the reconstruction error:

$$\mathcal{L}_{\text{DM}} = \mathbb{E}_{t,\boldsymbol{\epsilon}} \left[w(\lambda_t) \cdot \|\mathbf{z}_0 - \hat{\mathbf{z}}_0(\mathbf{z}_t, t, \mathbf{x}_{\text{pref}})\|_2^2 \right] \tag{3}$$

where $w(\lambda_t)$ is the lognormal weighting of Karras et al. (2022) of the log signal-to-noise ratio λ_t (Appendix F). The training loss is $\mathcal{L} = \mathcal{L}_{\text{LM}} + \beta \mathcal{L}_{\text{DM}}$, where β weights the diffusion loss. Because t is sampled during training, the decoder learns to generate the continuation from latents at every noise level. At high noise, the latent carries little information and the model must rely on the prefix; at low noise, it provides increasingly precise guidance. The model thus learns to use plans when they are informative while remaining robust to imperfect ones, and at inference we can choose the noise level of the plan it conditions on.

3.3 Inference

At test time, generation proceeds in three stages (Figure 1, right). We first process the prefix through the backbone. We then generate the latent plan by iterative denoising, starting from pure noise $\mathbf{z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and applying the denoiser for S steps to produce a latent $\mathbf{z}_0$ representing the intended output as a whole. Finally, we inject the plan after the prefix and decode autoregressively.

To strengthen the influence of the prefix on the generated plan, we use classifier-free guidance (Ho & Salimans, 2022). During training, we drop the prefix with probability $p_{\text{uncond}} = 0.1$, so the model also learns unconditional predictions. At inference, we interpolate:

$$\tilde{\boldsymbol{\epsilon}}_t = \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \varnothing) + w \cdot (\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{x}_{\text{pref}}) - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \varnothing)) \tag{4}$$

where $\boldsymbol{\epsilon}_\theta = (\mathbf{z}_t - \alpha_t \hat{\mathbf{z}}_0)/\sigma_t$ is the noise prediction and $w > 1$ increases the prefix's influence.

Implementation Details. The autoregressive backbone follows the Llama2 architecture (Touvron et al., 2023), with the diffusion experts described above. Implementation details, including how plans are sampled and decoded, and all hyperparameters are in Appendix G.

4 Experiments

We evaluate on tasks that stress-test three capabilities where autoregressive models struggle (planning ahead, global coherence, and creative exploration) and then on mathematical reasoning.

Baselines. We compare against autoregressive controls that each add one ingredient to AR_{Base}, which shares the backbone of AR w/ LDP, and match it on everything else. $\textit{AR}_{\text{Large}}$ adds capacity, with 2–4.5$\times$ the parameters. *Pause Token* (Goyal et al., 2024) adds computation without a planning objective: it inserts up to as many learnable tokens as our plan, at the same position, and trains them only through the language modeling loss. *Looped AR* (Saunshi et al., 2025) adds recurrent depth with shared weights, in two matched variants: *Looped* $\textit{AR}_{\text{Depth}}$ loops half of AR_{Base}'s layers twice, matching its depth and compute with half the parameters, and *Looped* $\textit{AR}_{\text{Param}}$ loops all of its layers twice, matching its parameters with twice the depth and compute. We evaluate these baselines on star graphs, Sudoku, and GSM8K; configurations and additional variants are in Appendix B.

4.1 Planning

Autoregressive models generate tokens sequentially, committing irrevocably to each choice before observing future context. This creates a fundamental challenge for tasks requiring foresight: the model must select the correct first token without knowing where the sequence will lead.

Star Graph Traversal. We adopt the star graph task from Bachmann & Nagarajan (2024). A star graph $G_{d,\ell}$ has a Start node at its center with d disjoint branches, each forming a path of ℓ nodes from Start to the end of the branch; one branch ends at the Goal. Figure 3 shows $G_{5,5}$, one of the configurations we evaluate. The input is the graph's edge list $(3, 5 \mid 31, 2 \mid \cdots)$ followed by the start/goal pair $(31, 10)$, and the model must generate the path from Start to Goal $(31, 2, 4, 9, 10)$. Only the first node after Start is difficult: node 31 has five neighbors, and choosing 2 requires tracing the path back from 10. This Difficult Token is the "indecipherable token" of Bachmann & Nagarajan (2024). The remaining Easy Tokens follow from the edge list alone: once 2 is given, 4 and then 9 are the only possible next nodes. Under teacher forcing, the ground-truth Difficult Token is always provided, so an autoregressive model achieves near-perfect accuracy by learning only the Easy Tokens. At inference, it must choose the Difficult Token itself, and without a plan it guesses among the Start node's d neighbors, yielding accuracy of approximately $1/d$. We evaluate on $G_{2,5}$, $G_{5,5}$, $G_{10,5}$, and $G_{20,5}$.

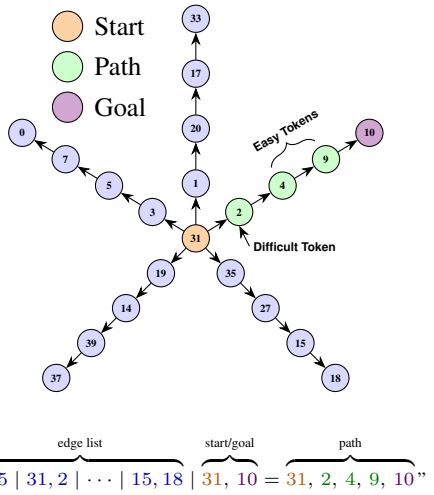
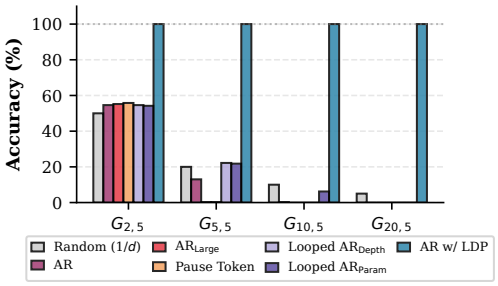
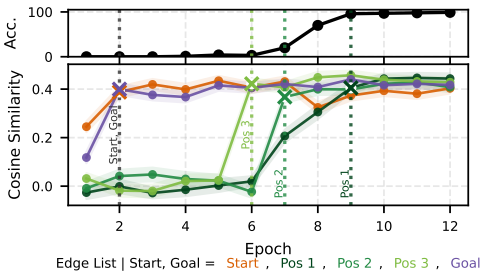


Figure 3: Star graph $G_{5,5}$. Given the edge list and start/goal, the model generates the Start-to-Goal path. Only the Difficult Token requires planning; the Easy Tokens follow from the edge list.



(a) Star graph accuracy. $G_{d,\ell}$ has d branches of ℓ nodes (Figure 3). Only AR w/ LDP rises meaningfully above chance $(1/d)$.



(b) Accuracy (top) and cosine similarity between the sampled latent and the target node embeddings (bottom) across training.

Analysis. AR w/ LDP achieves perfect accuracy across all star graph configurations (Figure 4a). The autoregressive baseline performs at chance level ($\sim$50%) on $G_{2,5}$ and degrades further as the number of branches d increases, falling to near-zero accuracy on $G_{10,5}$ and $G_{20,5}$. The other baselines fare no better: AR_{Large}, Pause Token, and both Looped AR variants remain near chance on every graph, so neither added capacity, extra computation, nor recurrent depth resolves the Difficult Token. The simplicity of this task enables detailed analysis of what the latent plan encodes. In Figure 4b, we track the cosine similarity between the sampled latent plan and the ground-truth embedding at each path position throughout training. A clear learning curriculum emerges. Initially, the model learns to encode the Start and Goal in its plan—information directly available in the input. However, knowing the endpoints alone is insufficient to determine the Difficult Token, and accuracy remains near zero.

The model then progressively learns to encode the intermediate positions, but in a striking order: Pos 3 (nearest the Goal), then Pos 2, and finally Pos 1, the Difficult Token (node 2 in Figure 3). Only when the plan encodes Pos 1 does accuracy jump to 100%. This backward curriculum reveals that AR w/ LDP learns to reason from the Goal back to the Start—precisely the strategy required to solve this task. An autoregressive model, forced to commit to Pos 1 before generating any subsequent tokens, has no mechanism to perform this backward reasoning.

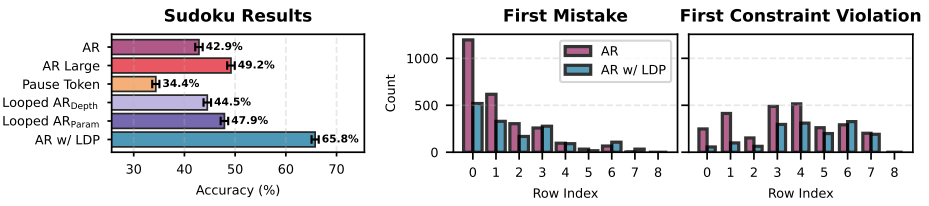


Figure 6: Sudoku results on 4,500 held-out test puzzles. Left: exact-match accuracy (± 1 s.e.). Center and right: for each failed puzzle, the row (0-indexed) of the first mistake (a digit that differs from the solution) and of the first constraint violation (a digit repeated in its row, column, or box).

4.2 Global Coherence

Many reasoning tasks require satisfying global constraints that span the output. Autoregressive models may commit to locally plausible choices that violate distant constraints, with no mechanism to backtrack. We study this with Sudoku, a canonical constraint satisfaction problem.

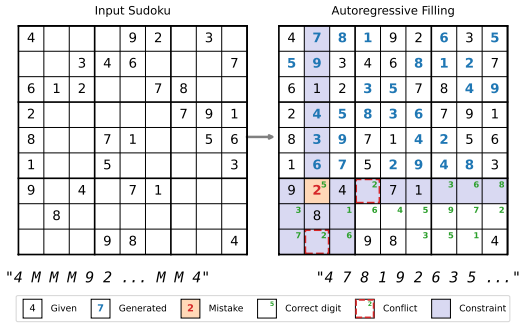


Figure 5: Sudoku as next-token prediction. The model fills the grid row by row. Its Mistake (2) repeats no digit among its observed Constraint cells, yet it rules out the 2 that both Conflict cells need, so the puzzle can no longer be completed.

Sudoku. Each row, column, and 3×3 box of a Sudoku grid must contain the digits 1–9 exactly once. We generate puzzles with 30–50 empty cells, each with a unique solution, following Stojanovski et al. (2025). As shown in Figure 5, the Input Sudoku is serialized row by row as 81 tokens, with a mask token (M) in each empty cell, and the model generates the complete grid in the same order (Autoregressive Filling). A digit can satisfy every rule visible when it is generated and still be wrong. In Figure 5, the model fills the first six rows correctly and then generates a 2 in the seventh row (the Mistake). No other 2 appears among its Constraint cells—its row, column, and box—so the choice looks valid, but the correct digit is 5. The solution needs a 2 in both Conflict cells, which the Mistake already rules out, so every completion must eventually break a rule.

Analysis. AR w/ LDP achieves higher accuracy than AR, with non-overlapping standard errors (Figure 6, left). Among the baselines, added capacity or computation helps modestly: AR_{Large} ($4.5\times$ the parameters) and Looped AR_{Param} (twice the depth and compute) each improve on AR by five to six points, Looped AR_{Depth} roughly matches AR, and Pause Token lowers accuracy. AR w/ LDP exceeds the best of them by a wide margin with far fewer parameters than AR_{Large}.

To see why, we separate the two kinds of failure in Figure 5: the first *mistake*, the first digit that deviates from the unique solution (like the 2), and the first *constraint violation*, the first digit that repeats one in its row, column, or box. If errors stemmed from poor constraint checking, the two would coincide. Instead, both models make their first mistake mostly in early rows, while violations cluster in later rows (Figure 6, center and right). The AR model's first mistake conflicts with its partial solution in only 14 of its 2,571 failed puzzles (0.5%), and the violation it forces arrives a median of 20 tokens later. Most errors are therefore *locally valid yet globally fatal*, so improvement must come from foresight rather than better constraint checking. AR w/ LDP makes its first mistake in early rows less often than AR, suggesting that latent planning provides this look-ahead.

4.3 Algorithmic Creativity

Problems typically admit multiple valid solutions, yet autoregressive models trained with maximum likelihood often favor solutions similar to those in the training distribution over the full solution space (McCoy et al., 2023). We probe this limitation with two graph discovery tasks, Sibling Discovery and Triangle Discovery (Nagarajan et al., 2025), illustrated in Figure 7. In both, relations between nodes are extracted from an underlying graph and partitioned into a *Train* split and a held-out *Test* split. Train alone contains enough information to recover the graph, so a model that learns the graph

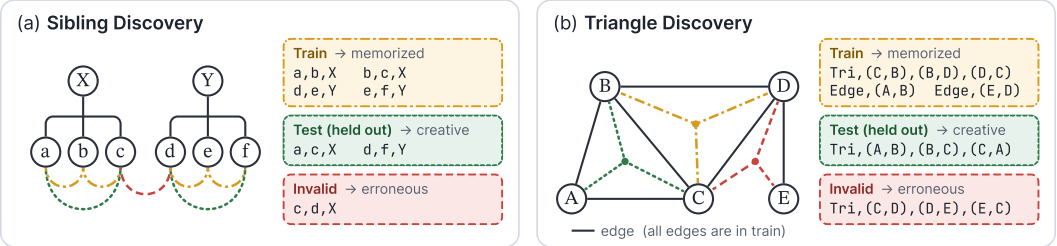


Figure 7: Algorithmic creativity tasks: (a) Sibling Discovery and (b) Triangle Discovery. A generation is *memorized* if it appears in the training set, *creative* if valid but held out, and *erroneous* if invalid.

can produce valid relations it has never seen. As the boxes beside each graph in Figure 7 show, this lets us label every generated relation precisely: *memorized* if it is in Train, *creative* if it is valid but held out in Test, and *erroneous* if it is invalid. A generation that repeats an earlier one in the same evaluation, up to permutation, is a *duplicate*.

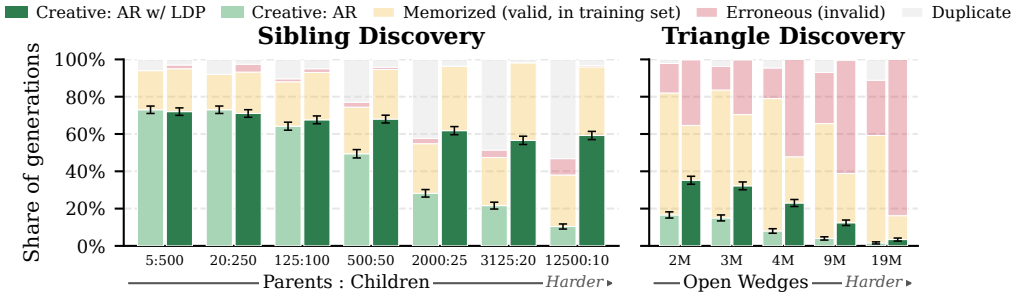


Figure 8: Algorithmic creativity results. Each bar splits 500 generations into creative, memorized, erroneous, and duplicate; in each pair the left bar is AR and the right bar AR w/ LDP. Error bars are binomial standard errors. Each run is shown at its checkpoint with the highest creativity.

Sibling Discovery. The underlying graph is bipartite: each of p parent nodes connects to its own set of c child nodes. Figure 7(a) shows $p = 2$ parents (`X`, `Y`) with $c = 3$ children each. A sibling relation names two children followed by their shared parent, so `a,c,X` states that `a` and `c` are siblings through `X`; the arcs beneath the children mark each relation, colored by split. The Train relations together cover every parent–child edge, so the graph can be recovered from them. A model that has learned the graph can then generate the held-out relations `a,c,X` and `d,f,Y` (creative), whereas `c,d,X` is erroneous because `c` and `d` do not share a parent. In our experiments, Train contains 20% of all relations, chosen so that every edge appears at least once, and the remaining 80% form Test. The task is trivial when $p = 1$ and grows more difficult as p increases. We sweep p:c (the x-axis of Figure 8, left) while fixing the total at 1.25M relations, isolating graph structure from dataset size.

AR w/ LDP is far more robust to sparsity (Figure 8, left). The two models are comparable when p is small and each child has many siblings, but as siblings become scarcer, AR's creative share falls to about 10%, mostly replaced by duplicates, while that of AR w/ LDP stays above 55%.

Triangle Discovery. The underlying graph is an Erdős–Rényi random graph (Erdős & Rényi, 1960) with n nodes and edge probability p (Figure 7(b)). Relations are either edges, such as `Edge,(A,B)`, or triangles, such as `Tri,(A,B),(B,C),(C,A)`, which are valid only if all three edges are in the graph and their node ordering forms a directed cycle. All edges are in Train (solid lines), while triangles are split between Train (20%) and Test (80%): `Tri,(C,B),(B,D),(D,C)` is memorized and `Tri,(A,B),(B,C),(C,A)` is creative. `Tri,(C,D),(D,E),(E,C)` is erroneous because `E`–`C` is not an edge; `C`–`D` and `D`–`E` form an *open wedge*, two edges that share a node but lack the closing third edge, a dead end. We prompt the model to generate triangles and sweep n and p while holding the expected number of triangles at 166K, so the number of open wedges (the x-axis of Figure 8, right) grows relative to the triangles. More open wedges mean more dead ends and a harder task.

AR w/ LDP generates more creative triangles than AR at every difficulty, roughly twice as many in the easiest settings (Figure 8, right). As open wedges become more common, the creative share of

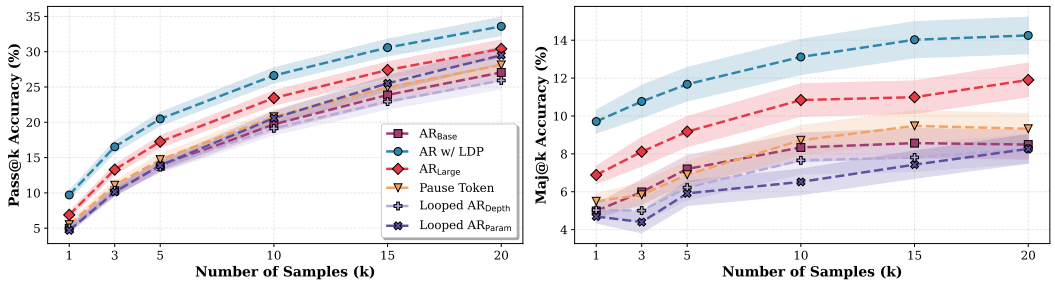


Figure 9: pass@k (left) and maj@k (right) on GSM8K (20 samples per problem; shading: ± 1 s.e.), each model at its best sampling setting (Appendix G). Other configurations are in Appendix B.

both models falls. The two models fail differently: most of AR's remaining generations reproduce training triangles (memorized), whereas those of AR w/ LDP are increasingly invalid (erroneous).

4.4 Mathematical Reasoning

TinyGSM. We test whether these gains transfer to mathematical reasoning with TinyGSM (Liu et al., 2023), a synthetic dataset of grade-school math problems paired with Python solutions. We train all models on 100k TinyGSM samples and evaluate on GSM8K (Cobbe et al., 2021) by executing the generated code. To avoid leakage from large-scale pretraining, the ModernBERT encoder underlying our autoencoder (Meshchaninov et al., 2025) is trained from scratch on the same data (Appendix E).

Results. AR w/ LDP lies above every baseline at every k on both pass@k and maj@k (Figure 9), and no configuration in Table 2 matches it on pass@1, pass@20, or maj@20. The strongest baseline, AR_{Large}, has three times the parameters. Extra computation without a planning objective does not help: Pause Token stays close to AR_{Base}, and so do both Looped AR variants. The gains are also cheaper than scaling: at the 10 denoising steps we report, AR w/ LDP is 13% faster than AR_{Large} (Figure 11). Removing the diffusion experts lowers every metric, but the ablated model still outperforms every baseline on pass@1 and pass@20 and ties AR_{Large} on maj@20 (Table 1).

Quality and Diversity. Why does a latent plan help where extra parameters and computation do not? An autoregressive model's samples can differ only in the tokens it samples, so its diversity comes from decoding controls such as temperature and top-p, which trade single-shot quality against diversity. Figure 10a sweeps both: along each AR frontier, gaining coverage (pass@20) costs single-shot accuracy (pass@1), while every guidance weight w of AR w/ LDP beats both frontiers on both axes. Panel (b) separates the plan's contribution from the decoder's. Decoded greedily, AR_{Base} is deterministic, so its pass@20 equals its pass@1. AR w/ LDP also decodes greedily, yet its pass@20 far exceeds its pass@1 because each sample starts from a freshly sampled plan. Replacing the plan with pure noise (no plan) collapses most of this coverage, while pass@1 barely changes and stays well above AR_{Base}: joint training improves the decoder itself, the plan supplies the coverage, and sampling tokens on top of the plan adds little. Panels (c) and (d) show what this buys. Coverage rises with the number of distinct answers per problem for every model, but at equal diversity AR w/ LDP reaches higher coverage (c). AR also pays for its diversity in broken programs: beyond roughly nine distinct answers per problem, its fraction of valid programs drops sharply, while that of AR w/ LDP stays flat at even higher diversity (d). Because the plan supplies the variation, the decoder can stay greedy and spends no token-level noise on arithmetic or syntax.

Table 1: AR w/ LDP on GSM8K with and without the diffusion experts (20 samples; s.e. ≤ 1.3).

AR w/ LDP	pass@1	pass@5	pass@20	maj@20
w/o experts	8.0	18.7	32.3	11.9
w/ experts	**9.7**	**20.5**	**33.6**	**14.3**

Inference-Time Settings. AR w/ LDP is forgiving of its inference settings. Because the decoder is trained on plans at every noise level (subsection 3.2), it can decode from any point in the diffusion process. We index this point by the noise level σ^2 of the plan the decoder conditions on, from the fully denoised plan ($\sigma^2{=}0$) to pure noise ($\sigma^2{=}1$, the no-plan setting in Figure 10b). Any level short of pure noise leaves pass@1 and pass@20 nearly unchanged. At each noise level the decoder follows the specific plan it is given: the solution is closer to its own plan than to other plans for the same problem, and this alignment strengthens as the noise level decreases (Appendix D). This shows that the plan steers decoder

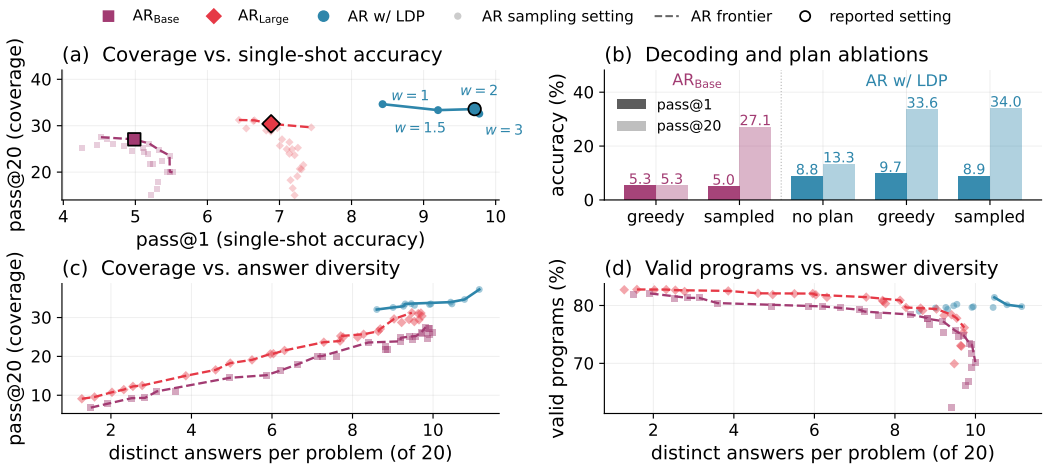


Figure 10: Quality and diversity on GSM8K (20 samples per problem). **(a)** Faint points are AR sampling settings, dashed lines their frontiers; outlined markers mark reported settings. **(b)** Greedy and sampled decoding at the reported settings; "no plan" decodes AR w/ LDP greedily from pure noise (σ^2=1). **(c, d)** Each point is one decoding setting (Appendix G); lines trace frontiers.

generation. Coverage saturates by 5–10 denoising steps, and every guidance weight from 1 to 3 beats all baselines on both pass@1 and pass@20 (Appendix C).

5 Related Work

Autoregressive models fail on algorithmic tasks that require foresight (Bachmann & Nagarajan, 2024), suffer from the "reversal curse" (Berglund et al., 2024), and favor solutions similar to their training examples over the full solution space (McCoy et al., 2023; Nagarajan et al., 2025). Diffusion has been adapted to language by discrete corruption of tokens (Hoogeboom et al., 2021; Austin et al., 2021; Hoogeboom et al., 2022; Lou et al., 2024; Sahoo et al., 2024) and by continuous diffusion over word embeddings (Li et al., 2022; Gulrajani & Hashimoto, 2023) or latent features (Lovelace et al., 2023; Zhang et al., 2023; Lovelace et al., 2025). Recently, Kang et al. (2026) use latent diffusion to generate latent thoughts as a post-training method; we instead integrate latent planning directly into pre-training. Chain-of-thought prompting (Wei et al., 2022) and reinforcement learning with verifiable rewards (Guo et al., 2025) elicit step-by-step reasoning in natural language. Other work adds latent computation without an explicit planning objective: pause tokens (Goyal et al., 2024), looped transformers (Saunshi et al., 2025), and recurrence over hidden states (Hao et al., 2024; Geiping et al., 2025). We compare against the first two in section 4. Transfusion (Zhou et al., 2024) and related models (Wang et al., 2024; Team, 2024; Liang et al., 2024) train one transformer for text and continuous image generation; we use this hybrid design for language reasoning.

6 Limitations and Conclusion

Limitations. Our controlled experiments are at modest scale; how the benefits of latent planning grow with model capacity and data remains open. We also study only pre-training; post-training hybrid architectures and combining latent planning with chain-of-thought reasoning are future work.

Conclusion. We demonstrate that latent diffusion can address fundamental limitations of autoregressive language models. By jointly training next-token prediction with diffusion over latent representations of the future, models learn to plan before committing to discrete tokens. Our hybrid architecture substantially outperforms autoregressive baselines on synthetic tasks isolating planning, global coherence, and creative exploration. On mathematical reasoning, it lets a smaller model exceed an AR model with nearly three times its active parameters at every sample budget, drawing its diversity from sampled plans rather than from degraded decoding. Latent planning offers a promising path toward more coherent, creative, and capable reasoning that complements autoregressive generation.

AI USE STATEMENT

We used large language model assistants for code development, analysis and plotting scripts, and editing of the manuscript text. All research ideas, experimental design, and scientific claims are the authors' own, and the authors verified all results reported in the paper.

REPRODUCIBILITY STATEMENT

We describe the model architecture and training procedure in subsection 3.2 and the surrounding section, and report implementation details and all hyperparameters in Appendix G, including how the decoding settings of every model were chosen. The text autoencoder and diffusion formulation are detailed in Appendix E and Appendix F. TinyGSM and GSM8K are publicly available, and we describe how the star graph, Sudoku, and sibling and triangle discovery datasets are generated. We will release our code upon acceptance.

REFERENCES

Emmanuel Abbe, Samy Bengio, Aryo Lotfi, Colin Sandon, and Omid Saremi. How far can transformers reason? the globality barrier and inductive scratchpad. *Advances in Neural Information Processing Systems*, 37:27850–27895, 2024.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=h7-XixPCAL.

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In *Forty-first International Conference on Machine Learning*, 2024.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: Llms trained on "a is b" fail to learn "b is a". In *The Twelfth International Conference on Learning Representations*, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Advances in Neural Information Processing Systems*, 38:57654–57689, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

Kanishk Gandhi, Ayush K Chakravarthy, Anikait Singh, Nathan Lile, and Noah Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STars. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=QGJ9ttXLTy.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach, 2025. URL https://arxiv.org/abs/2502.05171.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ph04CRkPdC.

Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-based diffusion language models. *arXiv preprint arXiv:2305.18619*, 2023.

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

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

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Towards non-autoregressive language models. *CoRR*, abs/2102.05379, 2021. URL https://arxiv.org/abs/2102.05379.

Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=Lm8T39vLDTE.

Emiel Hoogeboom, Thomas Mensink, Jonathan Heek, Kay Lamerigts, Ruiqi Gao, and Tim Salimans. Simpler diffusion (sid2): 1.5 fid on imagenet512 with pixel-space diffusion. *arXiv preprint arXiv:2410.19324*, 2024.

Liwei Jiang, Yuanjun Chai, Margaret Li, Mickel Liu, Raymond Fok, Nouha Dziri, Yulia Tsvetkov, Maarten Sap, and Yejin Choi. Artificial hivemind: The open-ended homogeneity of language models (and beyond). *Advances in Neural Information Processing Systems*, 38, 2026.

Haoqiang Kang, Yizhe Zhang, Nikki Kuang, Nicklas Majamaki, Navdeep Jaitly, Yian Ma, and Lianhui Qin. Ladir: Latent diffusion enhances llms for text reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 81705–81733, 2026.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 35:26565–26577, 2022.

Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and improving the training dynamics of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24174–24184, 2024.

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

Diederik P Kingma and Ruiqi Gao. Understanding diffusion objectives as the ELBO with simple data augmentation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=NnMEadcdyD.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of rlhf on llm generalisation and diversity. In *International Conference on Learning Representations*, volume 2024, pp. 20620–20653, 2024.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori Hashimoto. Diffusion-LM improves controllable text generation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=3s9IrEsjLyk.

Weixin Liang, Lili Yu, Liang Luo, Srinivasan Iyer, Ning Dong, Chunting Zhou, Gargi Ghosh, Mike Lewis, Wen-tau Yih, Luke Zettlemoyer, et al. Mixture-of-transformers: A sparse and scalable architecture for multi-modal foundation models. *arXiv preprint arXiv:2411.04996*, 2024.

Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. Tinygsm: achieving> 80% on gsm8k with small language models. *arXiv preprint arXiv:2312.09241*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion language modeling by estimating the ratios of the data distribution, 2024. URL https://openreview.net/forum?id=71mqtQdKB9.

Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Seo Shekhtman, and Kilian Q Weinberger. Latent diffusion for language generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=NKdtztladR.

Justin Lovelace, Christian K Belardi, Sofian Zalouk, Adhitya Polavaram, Srivatsa R Kundurthy, and Kilian Q Weinberger. Stop-think-autoregress: Language modeling with latent diffusion planning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=c05qIG1Z2B.

R Thomas McCoy, Shunyu Yao, Dan Friedman, Matthew Hardy, and Thomas L Griffiths. Embers of autoregression: Understanding large language models through the problem they are trained to solve. *arXiv preprint arXiv:2309.13638*, 2023.

Viacheslav Meshchaninov, Egor Chimbulatov, Alexander Shabalin, Aleksandr Abramov, and Dmitry Vetrov. Compressed and smooth latent space for text diffusion modeling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Rv6Lz84FlZ.

Vaishnavh Nagarajan, Chen Henry Wu, Charles Ding, and Aditi Raghunathan. Roll the dice & look before you leap: Going beyond the creative limits of next-token prediction. In *Forty-second International Conference on Machine Learning*, 2025.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

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, 2024.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TIdIXIpzhoI.

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

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2505.24760*, 2025.

Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*, 2024. doi: 10.48550/arXiv.2405.09818. URL https://github.com/facebookresearch/chameleon.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Xinlong Wang, Xiaosong Zhang, Zhengxiong Luo, Quan Sun, Yufeng Cui, Jinsheng Wang, Fan Zhang, Yueze Wang, Zhen Li, Qiying Yu, et al. Emu3: Next-token prediction is all you need. *arXiv preprint arXiv:2409.18869*, 2024.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, et al. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2526–2547, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yizhe Zhang, Jiatao Gu, Zhuofeng Wu, Shuangfei Zhai, Josh Susskind, and Navdeep Jaitly. Planner: Generating diversified paragraph via latent language diffusion model. *arXiv preprint arXiv:2306.02531*, 2023.

Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. *arXiv preprint arXiv:2408.11039*, 2024.

A Inference Efficiency

Setup. We time generation at batch size 1 on a single NVIDIA B300 with the same optimized inference stack for every model: a fixed-size KV cache allocated once, and each inner-loop step—an AR decoding step, or a full LDP denoising step including the unconditional classifier-free-guidance pass—captured as a single CUDA graph of compiled kernels. LDP encodes the prompt once and reuses its KV cache across all denoising steps and for decoding. This is exact because, under the blockwise mask, prompt tokens never attend to plan tokens. None of these optimizations changes the output (generations are identical to the unoptimized implementation). Latency is the mean over 300 GSM8K test problems, averaged over five interleaved repeats that agree to within 3.4%. Accuracy is measured on the full test set with 20 samples per problem.

Results. Planning adds a fixed cost of roughly 5 ms per problem plus 4.1 ms per denoising step (a linear fit is within 3% at every step count). Pass@1 is insensitive to the number of steps. Additional steps instead buy diversity, with Pass@20 saturating by 5–10 steps (Figure 11). At 5 steps, AR w/ LDP is 24% faster than AR Large while scoring 2.5 points higher on both Pass@1 and Pass@20. A single step already exceeds AR Large's Pass@1 at 30% lower latency, and costs 11% more than AR Base.

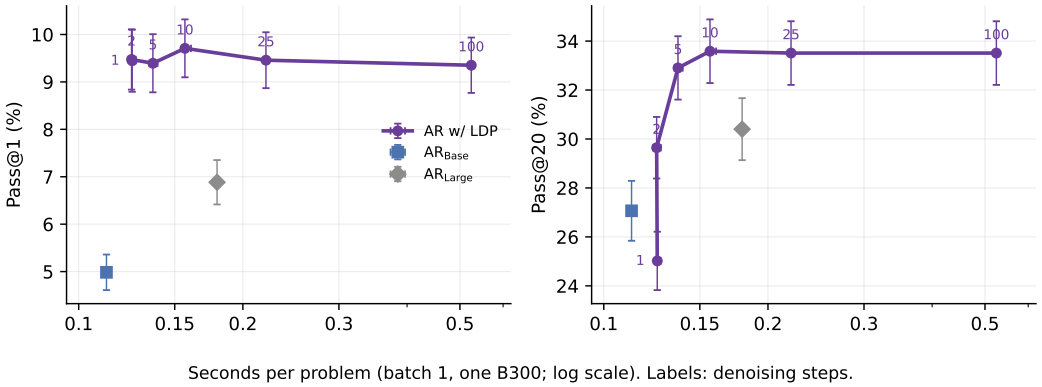


Figure 11: Quality versus latency on GSM8K at batch size 1, with the same optimized inference stack for all models. Point labels give the number of denoising steps; horizontal whiskers span five repeats. More steps leave Pass@1 unchanged and raise Pass@20 until 5–10 steps. At 5 steps, AR w/ LDP is faster than AR Large and better on both metrics.

B Additional Baselines

Every baseline is trained on each task with the same data, optimizer, learning-rate schedule, batch size, and number of epochs as AR_{Base}; only the architecture differs. AR_{Base} has 512 dimensions and 6 layers on star graphs and Sudoku and 8 layers on GSM8K, and AR_{Large} has 12 layers and 768 dimensions (512 on star graphs). Pause $\text{Token}_{\text{Base}}$ uses AR_{Base}'s architecture with as many pause tokens as the plan of AR w/ LDP on star graphs (4) and Sudoku (27); on GSM8K, where the plan has 32 tokens, it uses 16, which outperforms a 32-token variant. Looped AR_{Depth} and Looped AR_{Param} loop 3 and 6 layers twice on star graphs and Sudoku, and 4 and 8 layers twice on GSM8K, all at 512 dimensions. We also report Pause $\text{Token}_{\text{Large}}$ (AR_{Large}'s architecture, with 27 pause tokens on Sudoku and 32 on GSM8K) and a 32-token Pause $\text{Token}_{\text{Base}}$ on GSM8K. Table 2 lists every configuration.

Table 2: All baselines across tasks. Star graph: exact match on $G_{10,5}$. Sudoku: exact match on the 4,500 test puzzles not used for checkpoint selection ($\pm$1 s.e.). GSM8K: pass@k and maj@k over 20 samples ($\pm$1 s.e.), with every model at its best sampling setting (the 32-token Pause $\text{Token}_{\text{Base}}$ at T=0.8, top-p=0.95). Base models have 512 dimensions and 6 layers (star graph, Sudoku) or 8 layers (GSM8K); Large models have 12 layers and 768 dimensions (512 on star graphs). Looped AR_{Depth} loops half of the Base layers twice and Looped AR_{Param} all of them (3×2 and 6×2 on star graph and Sudoku, 4×2 and 8×2 on GSM8K). Pause $\text{Token}_{\text{Base}}$ uses as many pause tokens as the LDP plan on star graph and Sudoku (4 and 27) and 16 on GSM8K, where the plan has 32; the 32-token row matches the plan. Parameters: non-embedding, for the GSM8K models; for AR w/ LDP, active per token (total with both experts in parentheses; Table 6).

Model	Params	Star $G_{10,5}$	Sudoku	GSM8K pass@1	GSM8K pass@20	GSM8K maj@20
AR_{Base}	34M	0.2	42.9±0.7	5.0±0.4	27.1±1.2	8.5±0.8
Pause $\text{Token}_{\text{Base}}$	34M	0.0	34.4±0.7	5.5±0.4	28.1±1.2	9.3±0.8
Pause $\text{Token}_{\text{Base}}$ (32 tok)	34M	–	–	3.9±0.3	25.1±1.2	7.2±0.7
Looped AR_{Depth}	17M	0.0	44.5±0.7	5.0±0.4	25.9±1.2	8.3±0.8
Looped AR_{Param}	34M	6.2	47.9±0.7	4.7±0.3	29.5±1.3	8.3±0.8
AR_{Large}	113M	0.0	49.2±0.7	6.9±0.5	30.4±1.3	11.9±0.9
Pause $\text{Token}_{\text{Large}}$	113M	–	30.8±0.7	5.8±0.4	28.0±1.2	9.3±0.8
AR w/ LDP (no experts)	39M	–	–	8.0±0.5	32.3±1.3	11.9±0.9
AR w/ LDP	39M (64M)	**100.0**	**65.8±0.7**	**9.7±0.6**	**33.6±1.3**	**14.3±1.0**

C Inference-Time Settings

Figure 12 varies one inference setting of AR w/ LDP at a time on GSM8K. Accuracy is stable at every noise level short of pure noise, where pass@20 collapses; raising the guidance weight trades pass@20 for pass@1; and coverage saturates by 5–10 denoising steps. Every setting outperforms both autoregressive baselines except for pure-noise plans and one or two denoising steps.

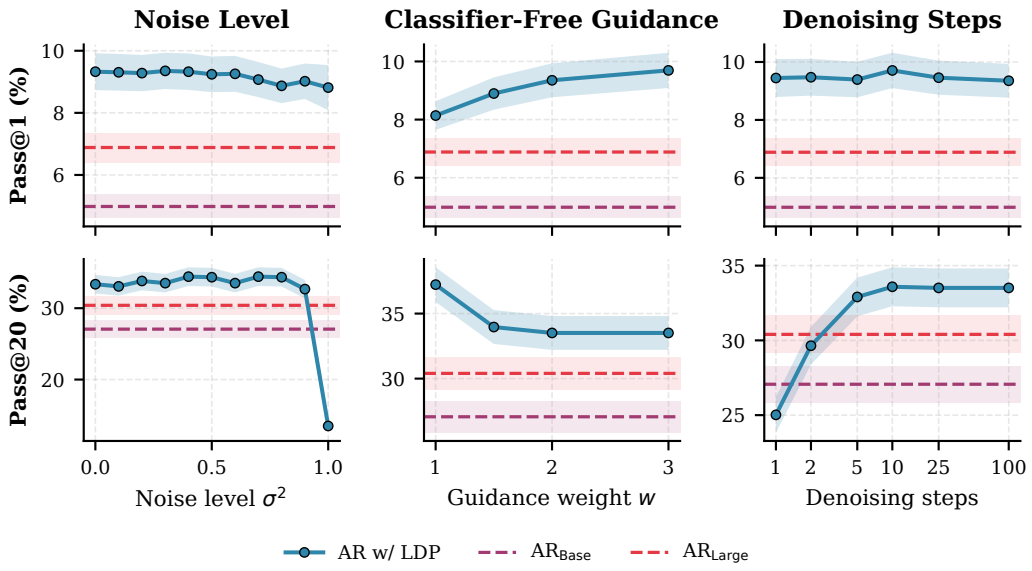


Figure 12: Inference-time ablations for AR w/ LDP on GSM8K: pass@1 (top) and pass@20 (bottom) against the noise level σ^2 applied to the sampled plan (guidance w=2, 100 denoising steps; left), the guidance weight w (σ^2=0.3, 100 steps; center), and the number of denoising steps (w=2, σ^2=0.3; right). Dashed lines are AR_{Base} and AR_{Large} at their best sampling settings; shading is ± 1 standard error.

D LATENT PLAN ANALYSIS

Plan–text alignment. To test whether decoded solutions follow their plans, we sample five plans for each of 200 GSM8K problems, decode a solution from each, and re-encode every solution into the latent space. Figure 13 (left) compares the similarity between each plan and the solution decoded from it (matched) with its similarity to the solutions decoded from the other plans for the same problem, and to solutions of other problems. At every noise level short of pure noise, a solution is closer to its own plan than to the other plans for its problem, so the decoder follows the specific plan it is given rather than only the problem. The gap narrows as noise increases and vanishes at $\sigma^2{=}1$, where the plan carries no information. The right panel shows the consequence: the sampled plans are equally diverse at every noise level, but once the plan is replaced by noise the decoded solutions become nearly identical, matching the collapse in coverage in Figure 10b.

Oracle plans. Table 3 encodes the ground-truth solution into a plan and decodes from it directly, bypassing diffusion sampling. The gap between sampled and oracle plans is the headroom a better diffusion model could recover; the gap between the oracle and 100% combines what the plan representation loses with errors the decoder makes even given the true plan.

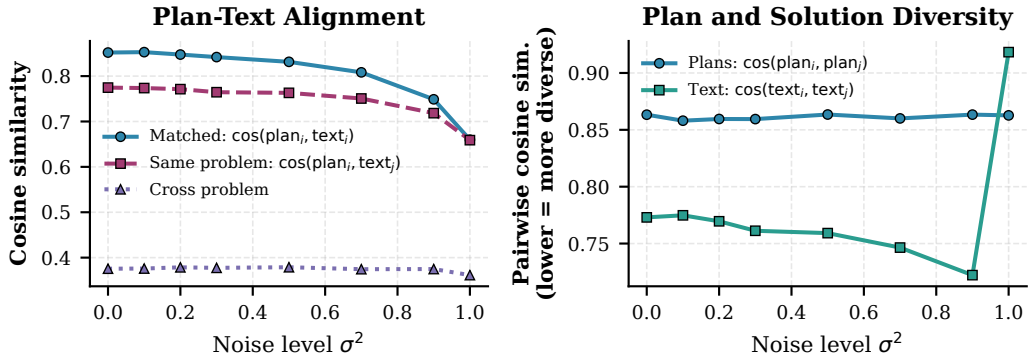


Figure 13: Latent plan analysis on GSM8K (200 problems, 5 sampled plans each; $w{=}2$, 10 denoising steps) against the noise level σ^2 of the plan the decoder conditions on. Left: cosine similarity between a sampled plan and the re-encoded solution decoded from it (matched), a solution decoded from another plan for the same problem, and solutions to other problems. Solutions follow their own plan at every noise level short of pure noise ($\sigma^2{=}1$). Right: pairwise similarity among the sampled plans and among the decoded solutions for the same problem (lower is more diverse). Without a plan, the solutions become nearly identical.

Table 3: Oracle plan quality. The oracle condition encodes the ground-truth continuation into a plan and feeds it directly to the AR decoder, bypassing diffusion sampling. The gap between the sampled and oracle plans is headroom attributable to the diffusion model; the gap between the oracle and 100% combines what the plan representation loses with errors the decoder makes even given the true plan. Sudoku numbers are on the 4,500 test puzzles not used for checkpoint selection; TinyGSM numbers are on a 500-example validation split with greedy decoding.

Task	Sampled plan	Oracle plan
Star graph (all configs)	100.0	100.0
Sudoku (exact match)	65.8	88.6
TinyGSM (execution acc.)	32.6	39.6

E AUTOENCODER

Text Encoder. To avoid leakage from large-scale pretraining, we train a ModernBERT encoder (Warner et al., 2025) from scratch on TinyGSM rather than using off-the-shelf pretrained models. The encoder is trained with masked language modeling (MLM) following standard practice (Devlin et al., 2019), where 15% of input tokens are randomly masked and the model learns to predict them. We train on the same 100k TinyGSM examples as the main models, for up to 1,000 epochs with early stopping, using a domain-specific BPE tokenizer with 4096 tokens. The resulting encoder provides semantically rich representations that serve as input to the compression stage. We present our encoder hyperparameters in Table 5. For each TinyGSM problem we stochastically sample either the question, solution, or concatenated question and solution.

AutoEncoder. We train a text autoencoder following Meshchaninov et al. (2025) to compress encoder features of the continuation into a low-dimensional latent space. The autoencoder maps from a variable-length input ($L \times d_{\text{enc}}$) to a fixed number of latent vectors ($k \times d_{\text{latent}}$). The autoencoder uses a Perceiver Resampler architecture (Alayrac et al., 2022) which consists of a stack of cross-attention layers that map from variable-length input ($L \times d_{\text{enc}}$) to a fixed number of latent vectors ($k \times d_{\text{latent}}$), and a symmetric decoder maps back to the original length for reconstruction. The autoencoder is trained with both token-level reconstruction loss (cross-entropy) and feature-level alignment (MSE against the frozen encoder representations), along with regularizations (e.g. dropout) that encourage a smooth latent space suitable for diffusion. During training of the main model, we freeze the autoencoder and perform diffusion in its latent space. We present our autoencoder hyperparameters in Table 4 and refer the reader to Meshchaninov et al. (2025) for further details.

Table 4: Autoencoder hyperparameters.

Hyperparameter	**Value**
Architecture	
Number of latent vectors (k)	32
Latent dimension (d_{latent})	16
Number of layers	12
Number of attention heads	8
Dropout	0.1
Encoder	
Base model	ModernBERT (Warner et al., 2025)
Max sequence length (L)	512
Regularization	
Random masking rate	0.3
Gaussian noise δ	0.3
Latent dropout rate	0.1
MSE loss weight (λ_{MSE})	0.1
Training	
Optimizer	AdamW
Learning rate	5×10^{-4}
Minimum learning rate	5×10^{-5}
Learning rate schedule	Cosine
Warmup steps	1000
Weight decay	0.01
Gradient clipping	1.0
Batch size	256
Max epochs	1000
Precision	bfloat16

Table 5: Text encoder (ModernBERT) hyperparameters.

Hyperparameter	**Value**
Architecture	ModernBERT (Warner et al., 2025)
Number of layers	8
Hidden dimension (d_{enc})	512
Number of attention heads	8
Max sequence length	512
Dropout	0.1
Masked Language Modeling	
Masking probability	0.15
Training	
Optimizer	AdamW
Learning rate	2×10^{-4}
Minimum learning rate	2×10^{-5}
Learning rate schedule	Cosine
Warmup steps	1000
Weight decay	0.01
Gradient clipping	1.0
Batch size	512
Max epochs	1000
Precision	bfloat16
Data	
Training examples	100k
Test examples	10K
Tokenizer	BPE (domain-specific)

F Diffusion

Training. The network predicts the clean latent ($\mathbf{x}$-prediction) on star graphs, Sudoku, and Sibling Discovery, and the velocity ($\mathbf{v}$-prediction (Salimans & Ho, 2022)) on Triangle Discovery and GSM8K (Table 7). We employ a lognormal v-loss weighting with mean 0 and standard deviation 2.4 (Karras et al., 2022; Kingma & Gao, 2023) combined with the cosine noise schedule (Dhariwal & Nichol, 2021) during training. For TinyGSM we sample logSNR values from a normal distribution with mean 0 and standard deviation 5, clipped to $[-15, 15]$, and adopt the multi-task reweighting scheme from Karras et al. (2024). To implement this, we track EMAs of the loss across noise levels and divide the MSE of each sample by the EMA of the corresponding noise level.

Sampling. We sample plans with the DDPM sampler (Ho et al., 2020); Appendix G gives the number of steps, noise schedule, guidance weight, and decoding noise level for each task. On GSM8K, accuracy is insensitive to the number of steps beyond 5–10 (Figure 11).

G Implementation Details and Hyperparameters

Training. All models are trained from scratch with AdamW (Loshchilov & Hutter, 2019), a linear warmup followed by cosine decay of the learning rate from 5×10^{-4} to 1×10^{-5}, weight decay 0.01, gradient clipping at 1.0, and bfloat16 precision; on GSM8K we also keep an exponential moving average of the weights (decay 0.99, updated every 10 steps after step 1,000). Table 7 lists the settings that differ across tasks. Every baseline on a task uses the same data, batch size, schedule, and maximum number of epochs as AR w/ LDP, except on GSM8K, where we cap the autoregressive models at 20 epochs because they begin to overfit by then. We cap each run at the listed number of epochs and early stop all runs. For Sudoku, we select the epoch with the highest accuracy on the first 500 puzzles of the 5,000-puzzle test set and report exact match on the remaining 4,500. We use specialized tokenizers for the synthetic tasks and a BPE tokenizer with 4,096 tokens trained on TinyGSM.

Parameter counts. We count non-embedding parameters, excluding the token embeddings, the LM head, and the latent embedding layer, whose sizes depend on each task's vocabulary. Each token passes through only one of the two expert FFNs, so for AR w/ LDP we report *active* parameters (the backbone with one FFN per layer, plus the DiT layers) and, separately, the total with both experts (Table 6). On GSM8K, the frozen text encoder and autoencoder (150M parameters together) are not counted. The parameter counts in Table 2 are those of the GSM8K models.

Table 6: Non-embedding parameters (millions). For AR w/ LDP: active per token / total with both diffusion experts.

Model	Star graph	Sudoku	GSM8K	Sibling	Triangle
AR_{Base}	25.2	25.2	33.6	14.2	14.2
Pause $\text{Token}_{\text{Base}}$	25.2	25.2	33.6	–	–
Looped AR_{Depth}	12.6	12.6	16.8	–	–
Looped AR_{Param}	25.2	25.2	33.6	–	–
AR_{Large}	50.3	113.3	113.3	–	–
Pause $\text{Token}_{\text{Large}}$	–	113.3	113.3	–	–
AR w/ LDP (no experts)	–	–	39.1	–	–
AR w/ LDP	29.1 / 48.0	29.1 / 48.0	39.1 / 64.2	22.0 / 32.6	19.9 / 30.5

Latent plans. The latent plan of each task consists of one or more latent vectors (Table 7). Pooled token embeddings give a single latent vector on star graphs and in the creativity tasks, and 27 on Sudoku (strided pooling of the 81 cells); the GSM8K autoencoder produces 32. The input DiT layers expand each latent vector into one or more diffusion tokens, and the output DiT layers map the diffusion tokens back to the latent vectors, so the plan occupies 4 diffusion tokens on star graphs, 27 on Sudoku, 8 in Sibling Discovery, 2 in Triangle Discovery, and 32 on GSM8K.

Plan sampling and decoding. At inference, we sample the latent plan $\hat{\mathbf{z}}_0$ with the DDPM sampler (Ho et al., 2020) and then decode from a plan at noise level σ^2: we re-noise the sampled plan to $\mathbf{z} = \sqrt{1-\sigma^2}\,\hat{\mathbf{z}}_0 + \sigma\boldsymbol{\epsilon}$ and give the decoder $\mathbf{z}$ together with its noise level, as in training (subsection 3.2). $\sigma^2{=}0$ decodes from the sampled plan itself and $\sigma^2{=}1$ from pure noise. On the synthetic tasks, we sample plans with 50 steps of the cosine schedule, without guidance, and decode at $\sigma^2{=}0$. On GSM8K, we use 10 steps of the shifted cosine schedule ($s{=}1.5$) (Hoogeboom et al., 2024), guidance weight $w{=}2$, and $\sigma^2{=}0.3$. AR w/ LDP decodes tokens greedily on every task, so all variation across its samples comes from the plan. The autoregressive baselines decode greedily on star graphs and Sudoku, sample at temperature 1 in the creativity tasks, and sample with a tuned temperature and top-p on GSM8K (below).

Algorithmic creativity. For Sibling Discovery, the number of parents p and children per parent c are reported on the x-axis of Figure 8. For Triangle Discovery, we sweep the node count n and edge probability p over the pairs (3997, 0.025), (1999, 0.05), (1000, 0.1), (667, 0.15), and (572, 0.175). Sibling Discovery holds out 80% of the sibling pairs as Test and trains on the remaining 20%; Triangle Discovery keeps every edge in Train and holds out 80% of the triangles. To evaluate, we prompt the model with the task's query token (for Triangle Discovery, followed by the triangle marker) and classify 500 generations; we evaluate every 0.1 epochs for Sibling Discovery and every 10 epochs for Triangle Discovery, and report the best evaluation of each run. The autoregressive baseline has the same backbone as AR w/ LDP without the experts (14.2M parameters, Table 6).

GSM8K decoding. Every model generates 20 samples for each of the 1,319 GSM8K test problems (at most 512 new tokens). The autoregressive baselines sample with temperature T and nucleus (top-p) sampling. For every baseline we sweep $T \in \{0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$ and top-$p \in \{0.7, 0.8, 0.9, 0.95, 0.99\}$ (Figure 10a shows the sweep for AR_{Base} and AR_{Large}) and report each at the setting on its pass@1–pass@20 frontier with the highest maj@20: $T{=}0.9$, top-$p{=}0.9$ for AR_{Base}; $T{=}0.9$, top-$p{=}0.95$ for AR_{Large}; $T{=}1.0$, top-$p{=}0.8$ for Pause $\text{Token}_{\text{Base}}$; $T{=}0.7$, top-$p{=}0.9$ for Pause $\text{Token}_{\text{Large}}$; $T{=}0.7$, top-$p{=}0.99$ for Looped AR_{Depth}; and $T{=}0.7$, top-$p{=}0.95$ for Looped AR_{Param}. The additional 32-token Pause $\text{Token}_{\text{Base}}$ uses $T{=}0.8$, top-$p{=}0.95$. AR w/ LDP decodes greedily; unless stated otherwise it uses guidance weight $w{=}2$, 10 denoising steps, and noise level $\sigma^2{=}0.3$, the setting on its pass@1–pass@20 frontier with the highest maj@20, chosen

Table 7: Per-task settings of AR w/ LDP. Baselines share the data and training settings of their task (GSM8K: at most 20 epochs). Pause Token, Looped AR, and the Large models differ only in architecture (Appendix B).

	Star graph	Sudoku	Sibling	Triangle	GSM8K
Data and training					
Training examples	200k	200k	Train split	Train split	100k
Batch size	256	768	256	1024	256
Max epochs	100	1000	8	5000	100
Warmup steps	1000	1000	100	1000	1000
Backbone					
Layers / width / heads	6 / 512 / 8	6 / 512 / 8	6 / 384 / 6	6 / 384 / 6	8 / 512 / 8
Latent plan and diffusion					
Latent	pooled emb.	pooled emb.	pooled emb.	pooled emb.	autoencoder
Latent vectors $\times$ dimension	1×16	27×16	1×64	1×64	32×16
Diffusion tokens	4	27	8	2	32
Prediction target	$\mathbf{x}$	$\mathbf{x}$	$\mathbf{x}$	$\mathbf{v}$	$\mathbf{v}$
Diffusion loss weight β	5	1	1	1	2
DiT width	256	256	256	256	256
DiT layers (input / output)	2 / 2	2 / 2	4 / 4	3 / 3	3 / 3
Inference					
Sampling steps	50	50	50	50	10
Schedule shift s	–	–	–	–	1.5
Guidance weight w	–	–	–	–	2
Decoding noise level σ^2	0	0	0	0	0.3

by the same rule as the autoregressive baselines. Figure 10(c, d) plots every swept setting: the 30 temperature and top-p settings of each autoregressive model, and 12 settings of AR w/ LDP (guidance weight $w \in \{1, 1.5, 2, 3\} \times \{5, 10, 100\}$ denoising steps, greedy decoding, σ^2=0.3). Panel (a) omits the near-greedy autoregressive settings with pass@20 below 15%, which lie far below both frontiers. A problem's distinct answers are the unique executed answers among its 20 samples, and a program is valid if it executes and returns a number.

GSM8K metrics. A sample is correct if its program executes and returns the ground-truth answer. With n=20 samples per problem, of which c are correct, we estimate pass@k with the unbiased estimator $1 - \binom{n-c}{k} / \binom{n}{k}$ (Chen et al., 2021) and average it over problems. maj@k takes a majority vote over the executed answers of the first k samples (ties go to the earliest sample) and scores the winning answer. A vote over one sample is that sample, so maj@1 equals pass@1, and we report pass@1 for it. Standard errors are taken over problems.