Discrete Diffusion Models Exploit Asymmetry to Solve Lookahead Planning Tasks

Itamar Trainin
Hebrew University of Jerusalem
itamar.trainin@mail.huji.ac.il

Shauli Ravfogel
New York University
shauli.ravfogel@nyu.edu

Omri Abend
Hebrew University of Jerusalem
omri.abend@mail.huji.ac.il

Amir Feder
Hebrew University of Jerusalem
amir.feder@mail.huji.ac.il

Abstract

Discrete diffusion models are often said to outperform autoregressive models on planning tasks, but the specific mechanisms enabling this are not yet well understood. To explain why, we abstract the complexity of studying "planning" into graph traversal problems, which offer a minimal setup for evaluating language models' planning and lookahead capabilities. We find that non-autoregressive language models are able to leverage an inherent directional asymmetry in lookahead planning tasks to outperform the training efficiency of their autoregressive counterparts. Through a mechanistic study of the training and inference dynamics of autoregressive and non-autoregressive models, we discover that while forward traversal through junctions in the graph requires complex sequential planning, the reverse path remains deterministic. Consequently, non-autoregressive models naturally develop a fundamentally different solution strategy. Effectively, they learn a reverse-decoding pattern which reduces an ℓth-order lookahead problem to a sequence of simple 1st-order transitions. Latent-space analysis further confirms these divergent strategies by demonstrating the distinct internal representations formed during generation. Ultimately, these findings highlight the advantages of non-autoregressive modeling while clarifying the inherent planning limitations induced by autoregressive architectures.

1 Introduction

The ability to plan a valid path towards reaching a desired goal is a fundamental requirement for complex decision-based systems [34]. To successfully plan, an agent must maintain a model of potential state transitions, looking ahead to understand not just immediate effects but how current choices constrain future possibilities [13, 9]. This lookahead often creates a structural asymmetry. The forward planning requires resolving complex ambiguities at branching junctions where multiple valid choices diverge, while the reverse process of tracing from a known goal back to the start is junction-free and thus deterministic (see Fig. 1).

Consider an itinerary planning task where an agent must schedule stops between two cities. Planning in the forward direction means that choosing a stop along the way requires evaluating how that decision restricts all subsequent options. Therefore, an agent must compute the entire trajectory, ensuring the final destination remains reachable. Conversely, starting from the goal and working backward can dissolve this complexity. Although Autoregressive (AR) modeling is the current standard for Large Language Models (LLMs), a recent line of work suggests that the causal "next-token" training objective poses an inherent conflict with this lookahead requirement [1, 23, 38, 37].

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

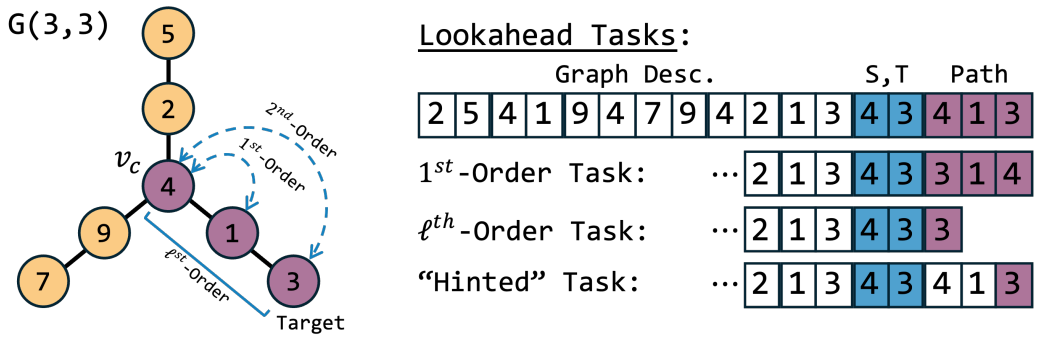

Figure 1: A Star-Path task graph (d=3 arms, l=3 vertices per arm), and its equivlent string representation for *Original* lookahead task, 1st*-Order* task where the path is decoded in reverse, ℓ^{th}*-Order* task where only the first and second vertices are predicted, and *"Hinted"* task where the true first and second vertices are provided in context. Purple indicates tokens predicted at test time.

Limited by their causal decoding direction, AR sequential models are unable to exploit the asymmetry in the lookahead task, forcing such model families to commit to a decision at a junction before the goal is fully processed. To mitigate this, AR models often rely on generating intermediate reasoning steps, effectively using their generated context as a "scratchpad" [25] to explicitly compute the necessary trajectory before yielding a final answer. However, generating these intermediate steps is not an out-of-the-box capability inherent to the sequential architecture itself, but rather relies on dedicated training. Consequently, [1] confirmed empirically that without such explicit intermediate scaffolding, AR models, whether Transformer-based [35] or State-Space [18, 7], default to random guessing at critical junction vertices, yet succeed when the task is reversed (junction-free). This fundamental limitation motivates our investigation of whether Non-Autoregressive (NAR) generative models offer a more suitable paradigm.

We thus hypothesize that NAR models can reduce complex long-distance lookahead to 1st-order neighbor transitions by exploiting the asymmetry in directionality. We focus on Discrete Diffusion Models (dLLMs) [5, 6], a class of NAR models that generate complete sequences by iteratively refining from noise. The training paradigm of dLLMs forces them to learn a more complex class of problems [20], potentially improving their effectiveness on benchmarks that require planning abilities, as has been recently shown by [37, 23, 38].

Rigorously assessing planning and lookahead capabilities in LLMs is challenging due to their ill-defined nature and the inherent lack of experimental control when evaluating them at full scale [34]. Nonetheless, one promising framework is the Star-Path task [1], which gives a controlled perspective of planning as a graph-traversal problem. In this task, the model receives a textual graph description along with source and target vertices, and must produce the connecting path. By introducing junctions to the graph topology, the model cannot determine the correct arm from local information alone. As such, the task enables a controlled planning setup in which the mechanisms learned by each architecture can be isolated.

To explore this hypothesis, we present a study extending the Star-Path framework to observe how these two generative paradigms tackle graph traversal (see Fig.1). Our investigation reveals a striking disparity in sample efficiency, as graph complexity scales, AR models require exponentially more data to converge than their NAR counterparts. We conduct a mechanistic study to identify the reason for this fundamental divergence in learning. While AR models hit a bottleneck attempting to master ℓ^{th}-order lookahead and build global structural "depth-maps", NAR models exploit structural asymmetry through a reverse-decoding strategy. By working backward from the goal, they reduce the planning problem to simple 1st-order transitions and rely on local identity matching. Ultimately, this study demonstrates how the flexible bidirectional context of Discrete Diffusion models enables solution strategies that are not only fundamentally different but also efficient for lookahead planning[1].

[1]Code and data available on: https://github.com/itamartrainin/dllms-planning.git

2 Related Work

2.1 Star-Path: A Textual Graph Algorithm Problem

The Star-Path task represents a key subset of broader research evaluating LLMs through textually represented graphs [19, 39, 36, 10]. Specifically, it aims to isolate planning and reasoning mechanisms via a synthetic and controllable environment. Distinguished by its minimality, the task has emerged as a benchmark for highlighting the limitations of current architectures in sequential processing [1, 33, 32, 24, 40, 12].

Initially, [1] used the Star-Path task to suggest that autoregressive (AR) architectures are fundamentally unable to solve lookahead tasks. Yet, subsequent work has demonstrated that the task is nonetheless solvable using such architectures [11, 24, 12]. Other studies have utilized the Star-Path setting to investigate the learnability of search versus retrieval [33], the implicit encoding of geometrical relationships [24], and the computational complexity hierarchy of graph reasoning [32].

2.2 Non-Autoregressive Language Models

While the field has largely converged on AR models due to their simplicity and scalability [28, 29, 4], NAR paradigms have re-emerged as a compelling alternative [8, 31]. Most notable among these are Discrete Diffusion Language Models (dLLMs) [5, 6], which, unlike earlier NAR architectures, combine bidirectional context with robust generative capabilities. Recent scaling efforts demonstrate that dLLMs can achieve performance competitive with state-of-the-art AR baselines at comparable costs [14, 23, 38].

Operating under a distinct training-inference paradigm, dLLMs forgo a single left-to-right factorization. Instead, they learn to denoise partially corrupted sequences in parallel, capturing a broad family of conditional distributions. Generation is then facilitated through an iterative denoising process that progressively reconstructs the sequence from noise [15, 20, 2]. Consequently, dLLMs exhibit superior performance on complex planning tasks [37]. Yet, the underlying mechanisms that drive this remain largely underexplored.

3 Method

To overcome the challenges of evaluating lookahead planning in full-scale language models, we utilize a minimal setup where the lookahead mechanism can be studied under control. Consider a graph $G := (V, E)$ and two vertices $v_c, v_t \in V$. Identifying a valid path traversing between these vertices requires a fundamental understanding of the graph's structure, including the possible transitions and a strategy for composing them successfully. Building upon recent research employing graph structures to analyze the emergence of algorithmic solutions during LLM training [1, 32, 33, 24], we leverage this setup to compare how AR and NAR models develop lookahead mechanisms by introducing junctions that obscure the direct path between vertices.

We train AR and NAR models by presenting them with arbitrary graph structures containing branching junctions, along with source and target vertices. The models are tasked with predicting the solution as a sequence of vertices forming the connecting path. By fully randomizing both the graph structure and the source-target pairs, we ensure the models cannot rely on memorization, forcing them to learn generalizable traversal strategies. Representing these graphs as tokenized strings maps the traversal problem into the sequential domain, providing an ideal testbed for investigating how different decoding paradigms handle long-distance dependencies and lookahead requirements. While many graph structures comply with this broad definition, we simplify our experimental setting to a star-shaped graph [1], a minimal graph structure containing a single decision point. This reduces the task to a precise decision point, allowing us to study the learned lookahead mechanisms in isolation. This setup naturally scales, as any junction-based graph can be represented as a composition of star graphs. Consequently, our primary investigation focuses on the star-graph setting, while more intricate composite structures are explored in Appendix A.4.

3.1 The Star-Path Task

A single Star-Path problem (an example in the data) is defined as a triplet consisting of a *graph description*, a *source-target vertex pair*, and the *unique connecting path*. As depicted in Fig.1, a Star-Graph $G(d, l) := (V_G, E_G)$ is a graph whose vertices (V_G) and edges (E_G) form a star configuration, featuring $d \in \mathbb{N}$ arms extending from a single center vertex $v_c \in \mathbb{V}$, each of length l. Consequently, the set of vertices is defined to be $V_G := \{v_c, v_{1,1}, ..., v_{1,l-1}, ..., v_{d-1,l-1}\} \subseteq \mathbb{V}$ so that $|V_G| = 1 + d(l-1)$. We define $\mathbb{V}$ as the global pool of allocatable vertices such that $|\mathbb{V}| = N$. The vertices on the graph are connected into a star shape by the set of edges $E_G := \{(v_{i,j-1}, v_{i,j}) \mid i \in [0, d),\ j \in [1, l), v_{\cdot,0} := v_c,\ v_{i,j} \in V_G\}$. The source-target pair is defined as $\mathcal{T} := (v_c, v_{t,l-1})$, where $v_{t,l-1}$ is a leaf vertex terminating the t-th arm $t \in [0, d)$. Finally, the connecting path is defined to be the ordered (inclusive) sequence of vertices from the source to the target $\mathcal{P} := (v_{t,i} : i \in [1, l); v_{t,0} := v_c)$.

For fixed star parameters (d, l), the i-th datapoint is defined textually as the triplet $S := [E_{G_i} \| \mathcal{T}_i \| \mathcal{P}_i]$. The graph $G_i(d, l)$ is constructed by uniformly sampling vertices without replacement from $\mathbb{V}$, yielding an overall sequence length of $|S| = (2 \cdot |E_{G_i}|) + 2 + (d(l-1)+1)$. To minimize confounding factors, we do not introduce separator tokens to mark sequence attributes (e.g., boundaries between the edge list, the source–target pair, and the path). Within this setup, because the source vertex v_c is connected to d identical arms, the correct first transition to $v_{t,1}$ is highly ambiguous. To generate this single token, an autoregressive model must identify the target vertex $v_{t,l-1}$ within the provided graph structure, recognize which arm t it belongs to, and successfully uncover the $l-1$ edges back to the center. Thus, a seemingly simple next-token prediction step inherently requires an l-th order lookahead operation.

3.2 Variations of the Star-Path Task

To fully study the lookahead mechanism, we introduce three additional variations to the original Star-Path task, controllably isolating different traits of the task. All variations are depicted in Fig.1.

1st-Order Lookahead Task: Models are trained to generate the "reverse-path" leading from the leaf vertex back to the center vertex. Since the reverse direction is junction-free, only first-order transitions need to be acquired by the model.

ℓ^{th}-Order Task: Models are trained to predict only the second vertex in the forward path, which is the critical junction choice. Under this setting, the vertices on the path from the leaf backwards cannot be used by the model. This restriction prevents models from using backward tokens as a scratchpad.

"Hinted Task": Like the original task, models are trained to predict the forward-going path, but they are hinted with the first and second vertices. This way, the critical junction choice is eliminated, requiring the modeling of only first-order transitions.

3.3 Experimental Setup

A generative model $LM_\theta(\boldsymbol{x}_{\text{prefix}}) = P(\boldsymbol{x}_{\text{path}} | \boldsymbol{x}_{\text{prefix}})$ is trained to solve the Star-Path task as defined in §3.1, where the task sequence is provided as the *prefix* and the *path* is the predicted sequence representing the solution. To compare the behavior of AR and NAR models, we train equivalent models. For the NAR variant, we adopt the Discrete Diffusion modeling paradigm first introduced by [14]. To ensure equivalence, we use a shared transformer backbone based on the `DDiT` implementation described by [14, 22, 2]. For AR modeling, we modify this implementation to (1) use causal self-attention, (2) set the diffusion time embedding to a constant value $t = 0$, and (3) replace the diffusion training and sampling procedures with standard AR next-token training and decoding.

In our experiments, we utilize a decoder-only transformer architecture [35] scaled to match standard small LLMs. This design choice facilitates direct comparisons with real-world models while maintaining computational efficiency during training. The vocabulary is defined to be the enumeration of $\mathbb{V}$, alongside a `[MASK]` token. All sequences within each training setup are of a fixed length. Refer to appendix §A.1 for model specifications and computational setup.

For a given experiment, we fix graph parameters (e.g., d,l) and construct a single held-out test set consisting of $20K$ unique Star-Path problems, which is shared across all runs. To support extreme

training set size, the training examples are generated on-the-fly at train time by sampling from a generator function. We ensure no sample repetition and no overlap with the test set. For each run, we record the number of training examples required to reach convergence, and enforce a maximum training budget after which training is terminated. Because training samples are generated online, no two runs observe the same samples or sample order, which further reduces potential confounding effects.

3.4 Training paradigms

As we are interested in generating the solution sequence given the task prefix, two possible training regimes can be employed. In a *conditional training* approach, the model is conditioned on the prefix sequence and trained exclusively on the path tokens. Consequently, the loss on the prefix tokens is masked out, and gradients are computed only for the path tokens. Alternatively, in a *full-sequence training* approach, the model is trained to generate the entire sequence (both prefix and path). Here, gradients are computed over all tokens, additionally rewarding the model for learning to generate valid graph descriptions. This method closely resembles the pre-training process used for standard LLMs.

In this work, we use the full-sequence training regime, as we found that conditional training requires a significantly larger number of samples to reach convergence. We provide a detailed comparison of the two regimes in Appendix §A.3, including ablation results demonstrating that prefix gradients are critical for AR convergence. Across all experiments, at test time, we provide the ground-truth prefix and decode only the path tokens.

3.5 Success Criteria

While standard loss and perplexity criteria are used to train the model, our primary focus is to evaluate performance on the lookahead task itself. For this reason, we measure convergence using an *exact-match* metric, formally defined in Appendix A.2. A prediction is considered correct only if the entire predicted path matches the ground-truth path token-by-token. A model is considered to have converged when accuracy exceeds > 0.95 for $t = 10$ consecutive validation steps, or when the maximum budget is reached.

4 Findings

4.1 Discrepancy in Convergence Complexity

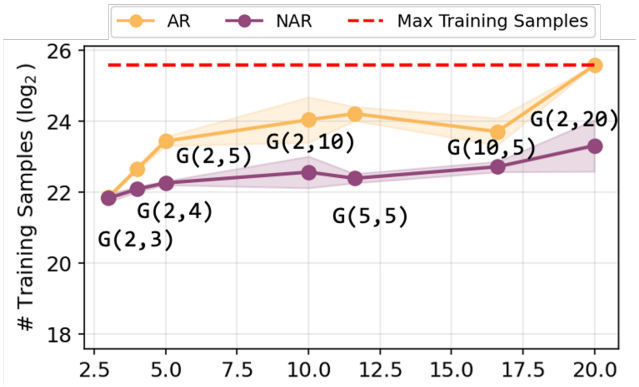

Figure 2: Comparison of AR and NAR convergence complexity as a function of Star-Path graph size. Axes (log_2 scale) show associative connections (x) and samples to convergence (y) when training AR and NAR models on the Star-Path Task. The red line (50M samples) marks the termination threshold for inconclusive convergence. Shaded areas indicate variance across 10 seeded runs.

Motivated by the asymmetry in the Star-Path task, we establish a baseline understanding of how AR and NAR models handle lookahead planning by evaluating their convergence behavior on the task. The results are reported in Fig.2. Although they are trained the same and are able to achieve

comparable performance, their sample complexity exhibits markedly different scaling behavior. For AR models, the required number of training samples grows rapidly. Conversely, NAR models exhibit a significantly flatter scaling trajectory. As the number of associative connections increases, the sample complexity gap between the paradigms widens substantially. Importantly, this discrepancy is robust to model scale, where significantly smaller and larger model sizes are evaluated (see Appendix A.7).

This naturally prompts the central hypothesis of our study of how NAR models learn to solve high-order planning lookahead tasks more efficiently. By attributing an associative memory structure to transformers [3, 30], we hypothesize that models learn to convert graph configurations into canonical representations for transition retrieval. The forward path from v_c to $v_{t,l-1}$ requires ℓ^{th}-order lookahead, but the reverse path is deterministic. NAR models can therefore adopt a reverse strategy requiring only 1st-order transitions. Since mastering higher-order transitions presupposes mastering simpler ones first, AR models face a compounding learning challenge that accounts for their steeper sample complexity.

To assess whether this advantage extends beyond the controlled Star-Path setting, we additionally evaluate full-scale NAR (dLLMs) and AR LLMs on established planning benchmarks, showing the NAR's superiority on such tasks and how scratchpad improves AR performance (Appendix A.9).

4.2 Mechanistic Adaptability

To validate the emergence of the reverse-decoding pattern in NAR models, we begin by investigating whether NAR models are capable of utilizing a flexible decoding strategy. While AR models are strictly restricted to the causal decoding direction, NAR models may implement alternative approaches. Fig. 3 visualizes the decoding process of our NAR model (dLLM; see §3.3) for a common setup of single-token unmasking at every inference step. Across multiple graph settings, we observe a consistent diagonal pattern moving from the bottom-right to the top-left, confirming that the NAR model adopts the reverse decoding strategy as hypothesized.

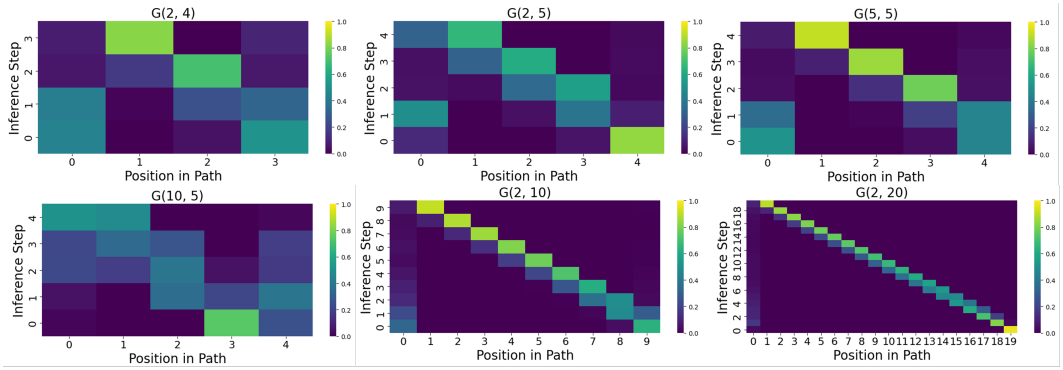

Figure 3: NAR models unmask tokens from goal to start, confirming reverse decoding. Each panel shows a different graph setting where the x-axis represents the vertex position in the path, and the y-axis represents the decoding step. Color intensity indicates the percentage of test examples (out of $20K$) where a token was unmasked at a given step.

Crucially, this adaptable behavior generalizes to both accelerated sampling, block-wise decoding, and more complex graph topologies, where the NAR model successfully identifies deterministic anchors to bypass complex planning. Fig. 4(a) shows the unmasking of multiple tokens simultaneously for faster decoding (top-$k = 2, 4, 8$ in Appendix A.5.1), where the model consistently utilizes a reverse-decoding pattern. Furthermore, when tested on the alternative "Two-Sided-Star" task (see task formulation in Appendix A.4), a topology designed with junctions at both the source and target, the NAR model dynamically expands outwards from a deterministic mid-path vertex in both directions (Fig. 4(b)). We further test this adaptability under block-wise generation; see Appendix A.5.2, Fig. 14.

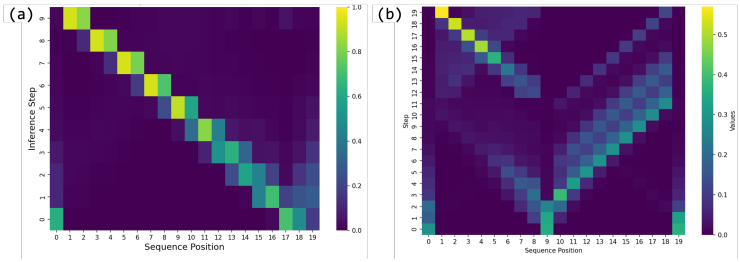

Figure 4: Decoding dynamics on the $G(2, 20)$ graph for (a) unmasking of $k = 2$ tokens at each decoding step and (b) NAR model (dLLM) trained on the Two-Sided-Star task. Color intensity indicates the percentage of test examples (out of $20K$) where a token was unmasked at a given step.

4.3 Exposing the Discrepancy in the Degree of Learned Transitions

Building upon this observed adaptive behavior, we design further experiments to explicitly expose the fundamental discrepancy between the complex l^{th}-order and simple 1st-order transitions acquired by AR and NAR models. By introducing the targeted modifications defined in §3.2 to the original Star-Path task, we control the specific types of transitions the models must acquire.

Fig. 5(a-c) compares sample complexity across these variations. The 1st-order task (a) enables AR models to use simple transitions, the ℓth-order task (b) forces NAR models to learn complex transitions, and the Hinted task (c) eliminates the junction decision for both. When both paradigms must learn the same transition order, their sample complexities match, confirming the impact of the order of transitions that need to be learned on the sample complexity. Furthermore, the discrepancy thus originates from the causal constraint forcing AR models to learn ℓ^{th}-order transitions.

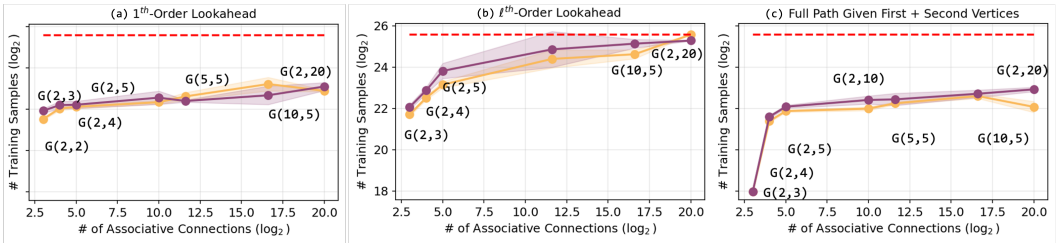

Figure 5: Comparison of 1st vs. ℓ^{th}-order lookahead across graph sizes. Axes (log_2 scale) show associative connections (x) and samples to convergence (y). Subplots represent: (a) 1st-Order, (b) ℓ^{th}-Order, and (c) Hinted tasks. The red line (50M samples) marks the termination threshold for inconclusive convergence. Shaded areas indicate variance across 10 seeded runs.

4.4 Hierarchy in Mastering Transitions

Once established that AR models must learn complex ℓ^{th}-order transitions, whereas NAR models only require simpler 1st-order transitions, we now deepen our investigation to understand why mastering these higher-order transitions demands a significantly larger sample complexity. Following the conclusions in [32], we claim that a transformer model must master the simpler lower-order transitions first, using them to uncover higher-order ones, thereby creating a learning hierarchy in the process of learning to lookahead. To demonstrate this, we focus on the training dynamics.

We show this hierarchical learning process by comparing the convergence dynamics. Fig. 6 depicts the *exact-match* training accuracy at equidistant intervals during the training process. The resulting learning trajectories reveal a stark contrast. While NAR models exhibit a single, rapid transition to perfect accuracy, AR models undergo two distinct "jumps" before converging. Crucially, the first AR jump coincides exactly with the NAR model's single jump to convergence. This phenomenon provides empirical support for our conjecture. We posit that this initial jump corresponds to the point at which both models successfully internalize the simpler 1st-order transitions. Because NAR models rely exclusively on these local mappings, this initial mastery is sufficient for immediate convergence.

In contrast, AR models must utilize these 1st-order transitions as a foundational prerequisite for deducing the more complex ℓ^{th}-order relations. Consequently, the AR models enter a prolonged plateau phase to internalize these higher-order dependencies. This extended phase directly accounts for their larger sample complexity, culminating in a second jump to convergence only once the entire hierarchy of transitions is mastered. In Appendix A.6, we examine the convergence of individual token positions, where the same behavior is observed. This qualitative learning hierarchy is also robust to model scale, as increasing model capacity shifts the transition points but does not eliminate the separation between the early acquisition of simpler transitions and the later AR convergence (Appendix A.7). Furthermore, in Appendix A.8, we experiment with a Mamba-based AR model, which exhibits smoother convergence and stronger encoding of graph structure while still showing a sample-efficiency gap relative to NAR.

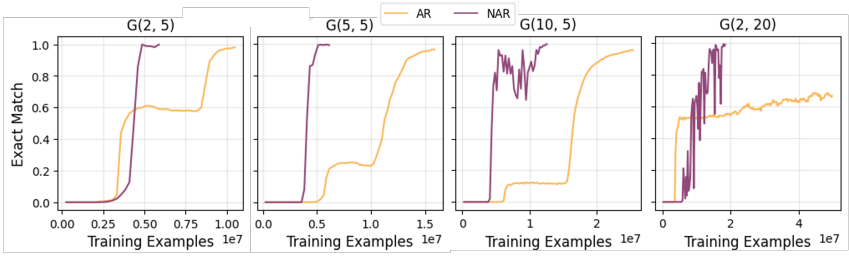

Figure 6: Train dynamics across varying graph configurations. AR models exhibit a two-phase convergence with a prolonged plateau, while NAR models converge in a single jump.

4.5 Latent Space Implications

To further substantiate our results, we investigate the latent space representations learned by both architectures. Our findings indicate that solving the task requires internalizing the dynamic configuration of the graph. We thus expect these latent representations to express a distinct structural understanding. To investigate this, we extract the hidden states generated at inference time by both paradigms across each transformer layer. Because our primary interest lies in how the models internalize the graph structure, we isolate our analysis to the representations of the graph description tokens (E_{G_i}). Specifically, we examine the ability of AR and NAR models to encode the topological depth of vertices (their distance from the center) and their capacity to encode the local "role" of a vertex, as conveyed through unique vocabulary identifiers (Vertex IDs).

We visualize the vertex representations (averaged over occurrences in description) by projecting them into 2D using Principal Component Analysis (PCA). We then color the resulting space to indicate the topological distance from the center and the corresponding Vertex ID. Fig. 7(a) and Fig. 21, 20 in the Appendix illustrate the layer-wise evolution of the latent space, colored using each schema. The visualizations reveal diverging strategies, the AR model strictly preserves a structured "depth-map" while rapidly dismantling its initial Vertex ID clustering, confirming its need to prioritize global dependencies over local identities for complex l^{th}-order forward planning. In contrast, the NAR model's initial depth separation quickly dissolves, yet it preserves Vertex ID clustering much deeper into the network. This diffuse topological representation, paired with strong local identity retention, perfectly aligns with its reverse-decoding strategy, demonstrating that NAR models sidestep the need for a persistent global hierarchy by relying on explicit token identities for simple 1st-order local transitions. This local strategy is further reinforced at the first layer, as can be seen in a zoom-in illustration (Fig. 7(a)). Unlike AR's topologically noisy Vertex ID clusters, NAR clusters display a clear depth gradient, indicating that bidirectional attention allows NAR models to "pre-compute" structural relationships immediately and bypass the AR's sequential learning bottleneck entirely.

We further evaluate these topological depth claims quantitatively by calculating the V-measure clustering scores on the original high-dimensional embeddings. Figure 7(b) tracks the V-measure across model layers. As shown, the representations generated by the AR model contain significantly more information regarding the distance of vertices from the center compared to NAR models. In stark contrast, the NAR model's score remains flat and low across all layers, quantitatively confirming our visual observation that its structural representation becomes diffuse and does not rely on a persistent global distance metric.

Ultimately, this provides further evidence for the distinct planning strategies adopted by each paradigm. While AR models are forced to construct and maintain a global structural hierarchy to navigate complex forward lookahead, NAR models exploit bidirectional context to rely entirely on local identity matching, seamlessly facilitating their highly efficient reverse-decoding approach.

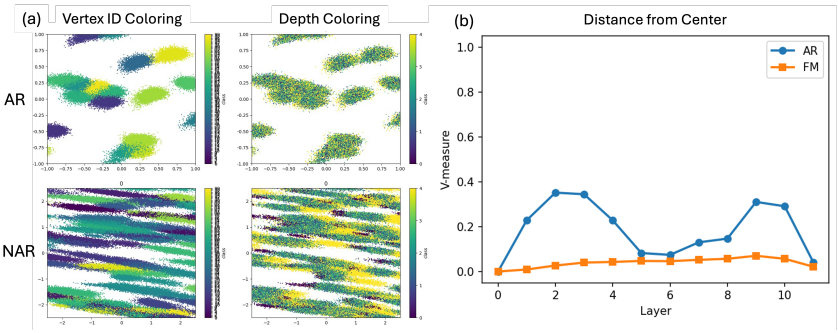

Figure 7: (a) Zoomed-in analysis of Layer 0 latent representation. While both models cluster by vertex ID, the NAR model organizes clusters into clear gradients based on topological depth. The AR model shows no such internal structure. (b) V-measure clustering scores for topological depth across model layers. The AR model actively encodes distance from the center, while the flat, low scores of the NAR model confirm it does not maintain a global structural hierarchy.

5 Discussion

We investigated the internal mechanisms of AR and NAR models on lookahead planning tasks. Contrary to previous claims that AR models are structurally incapable of such planning, we demonstrate that both paradigms can achieve perfect accuracy when provided with sufficient data and a structure-enforcing objective. However, they arrive at the solution through fundamentally different computational paths. A central insight from our work is the profound impact of directionality and asymmetry as an inherent phenomenon in planning problems. While forward planning often involves navigating complex, ambiguous branching pathways, the reverse trajectory is frequently deterministic.

Constrained by causal, left-to-right decoding, AR models are forced to navigate this structural ambiguity head-on, requiring them to master complex l^{th}-order lookahead patterns. Our analysis of training dynamics reveals that mastering this lookahead is a strictly hierarchical learning process. AR models hit a prolonged "learning bottleneck", establishing a foundation of simple 1st-order transitions before synthesizing them into global, high-order topological maps. Recognizing this inherent hierarchical dependency holds significant potential for improving the training dynamics of general LLMs.

In contrast, NAR models demonstrate remarkable generative flexibility by fully exploiting the task's structural asymmetry. By naturally adopting a "reverse-decoding" strategy, they effectively bypass the high-order planning challenge altogether, reducing it to a sequence of simple 1st-order neighbor transitions. While this mechanism allows NAR models to converge with exponentially fewer training examples, it reveals that the NAR models did not actually learn to "lookahead". Instead, they found a highly efficient computational shortcut. Therefore, while NAR modeling offers distinct advantages in flexibility and sample efficiency, we cannot confidently conclude that it possesses a fundamentally superior cognitive capacity for forward planning.

Moreover, this phenomenon highlights a critical vulnerability in modern LLM training. As standard training objectives optimize for likelihood rather than true causal reasoning, we have no guarantees regarding the actual internal algorithms a model develops. Models will exploit any available asymmetry or artifact to minimize loss, often resulting in behaviors that are empirically successful but mechanically misaligned with our intentions. As advanced architectures like dLLMs become adept at finding creative shortcuts, understanding *how* they solve problems becomes just as important as *whether* they solve them. Rigorous investigative studies, such as this work, linking latent mechanisms to observable behaviors, are therefore not merely academic exercises but rather indispensable for ensuring reliability and safety of future AI systems.

Limitations

While our study provides valuable mechanistic insights into how generative models approach planning tasks, our work has several limitations. First, our reliance on the minimal Star-Path framework rigorously isolates planning mechanisms, but due to its synthetic nature, it lacks the complexity, noise, and semantic richness of real-world tasks. As such, applying the conclusions in this work to actual LLMs requires further study. Secondly, because the remarkable sample efficiency of NAR models stems from exploiting structural asymmetries via reverse-decoding, it remains unclear if they hold an architectural advantage in environments with perfectly symmetrical ambiguity or where deterministic anchoring is impossible. This is especially true in the real world, where it is often difficult to identify the directionality and symmetry of problems, or whether the asymmetry is significant enough to enable a true advantage.

Disclosure of Funding

This work was supported by Israel Science Foundation (ISF) under the Grant No. 3123/25.

References

[1] Gregor Bachmann et al. The pitfalls of next-token prediction. In *International Conference on Machine Learning*, pages 2296–2318. PMLR, 2024.

[2] Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *arXiv preprint arXiv:2505.24857*, 2025.

[3] Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. Birth of a transformer: A memory viewpoint. *Advances in Neural Information Processing Systems*, 36: 1560–1588, 2023.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 2020. URL `https://arxiv.org/abs/2005.14165`.

[5] Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 2022.

[6] Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *arXiv preprint arXiv:2402.04997*, 2024.

[7] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 10041–10071. PMLR, 2024. URL `https://proceedings.mlr.press/v235/dao24a.html`.

[8] 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*, 2019. URL `https://arxiv.org/abs/1810.04805`.

[9] Lutfi Eren Erdogan, Hiroki Furuta, Sehoon Kim, Nicholas Lee, Suhong Moon, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Forty-second International Conference on Machine Learning*, 2025.

[10] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models, 2023.

[11] Arvid Frydenlund. The mystery of the pathological path-star task for language models, 2024.

[12] Arvid Frydenlund. Language models, graph searching, and supervision adulteration: When more supervision is less and how to make more more. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29011–29059, 2025.

[13] Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *Humanities and Social Sciences Communications*, 11(1):1–24, 2024.

[14] Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *Advances in Neural Information Processing Systems*, 37:133345–133385, 2024.

[15] Itai Gat, Heli Ben-Hamu, Marton Havasi, Daniel Haziza, Jeremy Reizenstein, Gabriel Synnaeve, David Lopez-Paz, Brian Karrer, and Yaron Lipman. Set block decoding is a language model inference accelerator. *arXiv preprint arXiv:2509.04185*, 2025.

[16] Gemma Team. Gemma 4 technical report, 2026. URL `https://arxiv.org/abs/2607.02770`.

[17] Google DeepMind. DiffusionGemma-26B-A4B-it model card. Hugging Face model card, 2026. URL `https://huggingface.co/google/diffusiongemma-26B-A4B-it`. Accessed 2026-09-11.

[18] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL `https://openreview.net/forum?id=tEYskw1VY2`.

[19] Bowen Jin et al. Large language models on graphs: A comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[20] Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025.

[21] Aakash Sunil Lahoti, Kevin Li, Berlin Chen, Caitlin Wang, Aviv Bick, Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In *International Conference on Learning Representations*, 2026. URL `https://proceedings.iclr.cc/paper_files/paper/2026/hash/8abd2043b71a074278d5f687947bff9c-Abstract-Conference.html`.

[22] Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky TQ Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

[23] 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. *arXiv preprint arXiv:2502.09992*, 2025.

[24] Shahriar Noroozizadeh, Vaishnavh Nagarajan, Elan Rosenfeld, and Sanjiv Kumar. Deep sequence models tend to memorize geometrically; it is unclear why. *arXiv preprint arXiv:2510.26745*, 2025.

[25] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[26] Qwen. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

[27] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI technical report, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[29] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[30] Adityanarayanan Radhakrishnan, Mikhail Belkin, and Caroline Uhler. Overparameterized neural networks implement associative memory. *Proceedings of the National Academy of Sciences*, 117(44):27162–27170, 2020.

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[32] Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. Understanding transformer reasoning capabilities via graph algorithms. *Advances in Neural Information Processing Systems*, 37:78320–78370, 2024.

[33] Abulhair Saparov, Srushti Ajay Pawar, Shreyas Pimpalgaonkar, Nitish Joshi, Richard Yuanzhe Pang, Vishakh Padmakumar, Mehran Kazemi, Najoung Kim, and He He. Transformers struggle to learn to search. In *The Thirteenth International Conference on Learning Representations*, 2025.

[34] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005, 2023.

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

[36] Heng Wang et al. Can language models solve graph problems in natural language? In *Advances in Neural Information Processing Systems*, volume 36, pages 30840–30861, 2023.

[37] Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. *arXiv preprint arXiv:2410.14157*, 2024.

[38] Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

[39] Ruosong Ye et al. Language is all a graph needs. In *Findings of the Association for Computational Linguistics: EACL 2024*, 2024.

[40] Yi Zhang et al. Unveiling transformers with lego: A synthetic reasoning task, 2022.

[41] Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. NATURAL PLAN: Benchmarking LLMs on natural language planning. *arXiv preprint arXiv:2406.04520*, 2024. URL https://arxiv.org/abs/2406.04520.

A Appendix

A.1 AR and NAR Implementation Specifications

Our implementation of an autoregressive and non-autoregressive model was based on the Flow-Matching repo[2]. To enable comparable AR and NAR modeling, we have modified the training and inference implementations to standard autoregressive procedures based on the same architecture. The following are the model parameters used for training the AR and NAR models:

Table 1: Configuration hyperparameters.

Parameter	**AR**	**NAR**
Causal Masking	✓	×
Autoregressive Decoding	✓	×
Batch Size	1024	
Max. Training Examples	50×10^6	
Optimizer	AdamW	
Learning Rate	3×10^{-4}	
β_1	0.9	
β_2	0.95	
ϵ	1×10^{-8}	
Weight Decay	0.03	
Warmup Steps	5×10^3	
Grad Clip	1.0	
$\eta_{\min}$ Ratio	0.01	
Accumulation Steps	1	
Loss Function	Cross Entropy	
Source Distribution	Masked Seq.	×
Scheduler Type	Polynomial	×
Hidden Size	384	
Cond. Dim.	4	
# Transformer Blocks	12	
# Attention Heads	6	
Dropout	0.1	

This study utilized approximately 500 GPU hours of compute time on NVIDIA L40 GPUs.

A.2 Success Criteria

In this study, we primarily focus on evaluating the performance of the model's ability to predict the path from the source to the target vertices. For this reason, we measure convergence using the *exact-match* metric. A prediction is considered correct if and only if the entire predicted path matches the ground-truth path token-by-token.

Formally, let $\tilde{\boldsymbol{x}}_{\text{path}} = LM_\theta(\boldsymbol{x}_{\text{prefix}})$. Then, for a test set $\mathbb{D}$,

$$\mathcal{A} := \frac{\sum_{(\boldsymbol{x}_{\text{prefix}}, \boldsymbol{x}_{\text{path}}) \in \mathbb{D}} \mathbf{1}\left\{\tilde{\boldsymbol{x}}^i_{\text{path}} = \boldsymbol{x}^i_{\text{path}} \ \forall\, i \in [0, l)\right\}}{|\mathbb{D}|} \tag{1}$$

where $\boldsymbol{x}^i$ denotes the i-th token in the sequence $\boldsymbol{x}$.

We use the validation set to evaluate *exact-match* accuracy at equidistant intervals during the training process. A model is considered to have converged when the exact-match accuracy exceeds > 0.95 for 10 consecutive evaluation steps, or when the maximum training budget (in examples) is reached.

[2]https://github.com/facebookresearch/flow_matching

A.3 Learnability of The Star-Path

Contrary to recent claims suggesting that AR models are fundamentally incapable of solving lookahead planning tasks [1], our results demonstrate that standard AR Transformers can learn the Star-Path task with perfect accuracy. We posit that the previously reported failures are not due to inherent structural limitations of the architecture, but rather stem from the optimization signal provided during training. Specifically, we identify that the learnability of the task heavily depends on whether the model is incentivized to internalize the graph structure.

To isolate this effect, we compare two training objectives. The standard approach models the conditional probability $p(\text{path}|\text{prefix})$, where the prefix (graph description and source-target pair) is provided as a prompt context without gradient propagation on its tokens. While theoretically sufficient, we find that this formulation renders the optimization landscape difficult to traverse, leading to slow or stalled convergence. In contrast, as can be seen in Fig.8, by enabling gradient propagation on the prefix and treating the graph description as part of the generative modeling task, we observe a dramatic acceleration in convergence. Although the precise dynamics of this pre-training-like phenomenon warrant further investigation, it suggests that forcing the model to explicitly model the graph structure rather than merely attending to it is crucial for acquiring the necessary lookahead mechanisms.

However, we observe that even with this optimized objective, fully mastering the task necessitates a substantial volume of training data. In our experiments, we capped the training budget at 50M sequences, an order of magnitude more than previous work (capped at 200K with multiple epochs). This indicates that while learnable, the task remains sample-inefficient for AR architectures. Furthermore, we observe that for graph configurations, even this extensive budget occasionally proved insufficient. Crucially, all models are trained for a single epoch over non-repeating data streams. This strict regime ensures the model cannot overfit to specific training instances, further establishing that our reported results reflect generalization rather than memorization.

Fig.8 illustrates the empirical impact of this training choice across varying graph complexities. We compare the Exact Match accuracy of AR models trained with and without prefix gradients on an equal number of examples (50M). The results reveal that models trained with prefix gradients (orange bars) were able to achieve near-perfect accuracy (> 0.95)[3] in most settings. Conversely, the baseline approach (pink bars) fails to converge. These findings demonstrate that the primary bottleneck in AR planning is not the search capacity, but the richness of the supervision signal provided during training.

Given the potential impact of these findings, we encourage further research to better understand the intricate dynamics of this experimental insight.

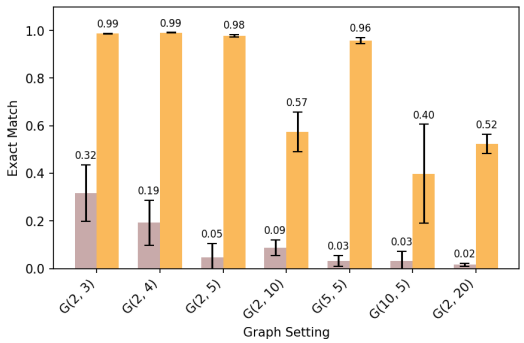

Figure 8: Comparison of AR models trained with (orange) and without (pink) gradients on the graph prefix across graph settings. All models were trained on up to 50M distinct training examples. Confidence intervals were computed based on 10 seeded runs.

[3]To reduce computational and environmental costs, runs maintaining a convergence score higher than 0.95 for 10 consecutive evaluation steps were terminated early.

A.4 Beyond Star-Graphs

In the main text (§3), we justified our focus on the minimal Star-Graph topology $G(d, l)$ by reasoning that any arbitrary graph structure containing branching junctions can be fundamentally represented as a composition of simple star graphs. The minimal setting allowed us to isolate and study the learned lookahead mechanisms with maximum precision. However, to rigorously demonstrate the generalization of our findings beyond a single junction, we introduce and analyze an extension of the Star-Path, the "Two-Sided-Star" use-case.

The Two-Sided-Star configuration is designed to eliminate any inherent directional bias in the task setup. In the original Star-Path task (§2.1), the reverse direction (from leaf to center) is naturally junction-free. To remove this asymmetry, we construct the graph structure illustrated in Fig. 9. This topology consists of two symmetric star-graph components (a "source-side" and a "target-side") joined by a shared central mid-path vertex (v_{mid}, introduced to make the path unique, shaded purple in the figure). The source vertex (v_s) and the target vertex (v_t) are placed as leaf nodes on opposing sides. Crucially, as Fig. 9 highlights, both v_s and v_t face branching junctions immediately upon traversal (vertices 34 and 7, respectively). By randomizing which arm on each side completes the full path, we create a task where both the forward and backward directions from the endpoints require resolving high-order lookahead to identify the unique connecting path through v_{mid}. Local information alone is insufficient from either endpoint.

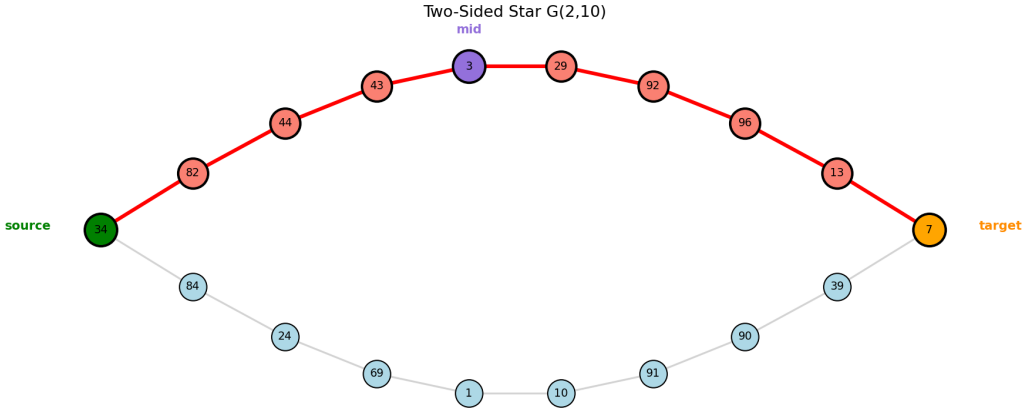

Figure 9: Structure of the "Two-Sided-Star" graph topology. Branching junctions exist adjacent to both the source (green) and target (orange) vertices, removing any trivial reverse-decoding advantage from the goal. The highlighted path (red) must be deduced, connecting source to target through a provided mid-path vertex (purple).

The expected behavior for a model truly exploiting structural asymmetry would be to identify the most deterministic part of the task and utilize that as a decoding anchor. In the Two-Sided-Star, while the endpoints are ambiguous, the path leading from the provided mid-path vertex (v_{mid}) outwards is deterministic. An optimal adaptive strategy would involve "anchoring" at v_{mid} and decoding sequentially towards both v_s and v_t using simple 1st-order transition matching, which is junction-free. To evaluate whether NAR models adopt this expected behavior, we trained a NAR model (dLLM) on this specific composite structure. Upon evaluating the trained model's performance, we found that it converges perfectly and exhibits the sample efficiency advantages observed in the single-junction case. We explicitly examined the decoding pattern of this model to uncover its underlying strategy. The result of this mechanistic deep-dive is presented in Fig. 10. The visualization confirms that the model implements the precisely hypothesized dynamic adaptive decoding mechanism. Across different configurations ($G(2, 10)$ and $G(2, 20)$), a strikingly consistent decoding pattern emerges.

The diagonal unmasking pattern indicates that the NAR model does not decode causally (left-to-right) or strictly backward (right-to-left). Instead, it dynamically expands outwards from the provided mid-path vertex towards both sides. This strategy brilliantly reduces the complex global planning problem into two parallel sets of simple, 1st-order deterministic neighbor transitions. This is functionally equivalent to the adaptive mechanism discovered in the original star-path, but here applied to a composite structure where simple reverse-decoding from the target was made impossible. This

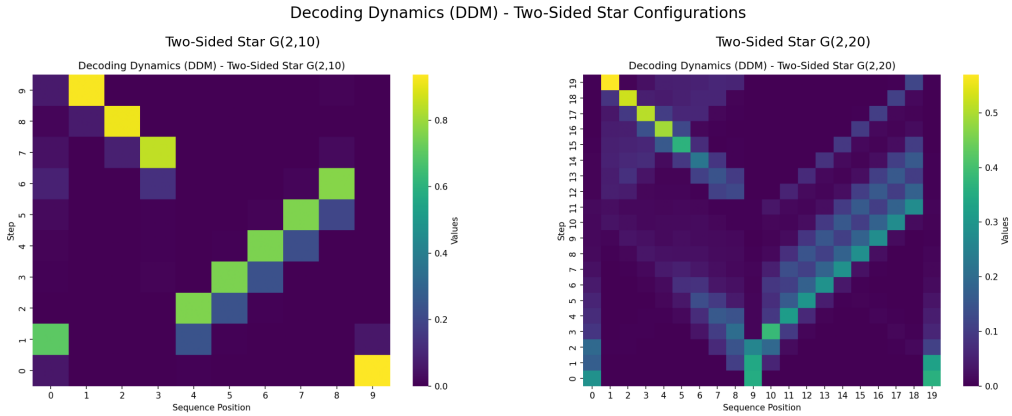

Figure 10: Decoding dynamics of a trained NAR model (dLLM) on Two-Sided-Star configurations. As unmasking proceeds over time (y-axis), the model prioritizes unmasking tokens moving outwards from the mid-path vertex towards both the target and source simultaneously. This adaptive, bidirectional expansion allows the model to solve the task using only simple 1st-order transitions, despite junctions being present from both ends.

finding strongly suggests that Discrete Diffusion models are not exploiting a simple directionality trick, but are dynamically adapting their generation order to find the most efficient computation path, a capability fundamentally unavailable to Autoregressive models forced into causal sequential constraints.

A.5 Mechanisitc Adaptability - Additional Results

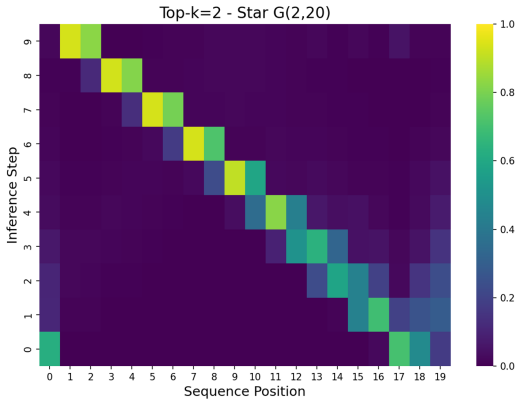

Figure 11: Decoding dynamics for $k = 2$ on the $G(2, 20)$ graph. Even when unmasking multiple tokens simultaneously (top-k logit strategy), the model consistently utilizes 1st-order connections in a reverse, non-autoregressive manner to solve the task, mirroring the baseline $k = 1$ strategy.

A.5.1 Accelerated Sampling

In this work, we initially focused on the standard inference setup for discrete diffusion models, unmasking a single token per inference step ($k = 1$). However, a major motivation for utilizing dLLMs is their capacity to significantly improve decoding speed by unmasking multiple tokens ($k > 1$) at each decoding step. To evaluate whether the models maintain their robustness and structural problem-solving behavior under these faster sampling strategies, we investigate the decoding dynamics when unmasking $k > 1$ tokens per step. Specifically, we apply a commonly used unmasking strategy where the top-k tokens are selected based on their highest predicted logit values. All resulting figures and metrics are computed over the validation set of size 20K graph configurations.

The results demonstrate that for smaller values of k, the model's fundamental strategy remains completely intact. Fig. 11 illustrates the decoding pattern for $k = 2$ on the $G(2, 20)$ topology. The visualization shows that even when unmasking multiple tokens simultaneously, the model consistently utilizes simple 1st-order connections in a non-autoregressive manner to solve the task. By successfully anchoring at the target and decoding backward, the model's behavior perfectly mirrors the $k = 1$ baseline.

However, as we push the limits of this accelerated sampling by increasing k to 4 and 8 (Fig. 12), this rigid reverse-decoding pattern begins to fracture. While the model still attempts to decode backwards from the goal, it simultaneously begins unmasking tokens in the forward direction. This dual-directional unmasking is accompanied by a significant degradation in path accuracy. As depicted in Fig. 13, while the $k = 1$ and $k = 2$ settings maintain near-perfect mean per-position accuracies of 0.9996 (min 0.999) and 0.9957 (min 0.987), respectively, increasing k leads to a severe performance drop. For $k = 4$, the overall accuracy falls to 0.7555, with intermediate sequence positions dropping to $\sim 0.45 - 0.50$ before recovering toward the end of the path. Under the extreme $k = 8$ setting, the model struggles drastically, achieving a mere 0.3555 accuracy. At this accelerated rate, the intermediate path positions plummet to an accuracy of roughly 0.04, successfully resolving only the immediate adjacent vertices (positions 0 and 19) alongside the tail end of the path, further demonstrating that local 1st-order transitions were successfully learned. Crucially, because the model fundamentally fails to solve the task at these higher k values, these fractured decoding patterns cannot be reliably used to either support or contradict our core claims regarding lookahead mechanisms. Nevertheless, the specific nature of this failure provides an interesting insight into the model's internal representations.

This breakdown reveals an intriguing nuance regarding the internal representations learned by the NAR model. The fact that the model attempts forward prediction for high k values, albeit unsuccessfully, implies that it did acquire some latent representation of higher-order lookahead (> 1) during training. However, this high-order understanding is fragile. Not only does it struggle to resolve higher-order transitions from the forward direction, but it also has a hard time accurately following 4 or 8 transitions backward in a single step. Without the step-by-step, deterministic grounding provided by sequential 1st-order reverse transitions, the model's internal representations are insufficiently precise to accurately bridge the long-distance dependencies required to solve the task.

We leave further investigation to future work.

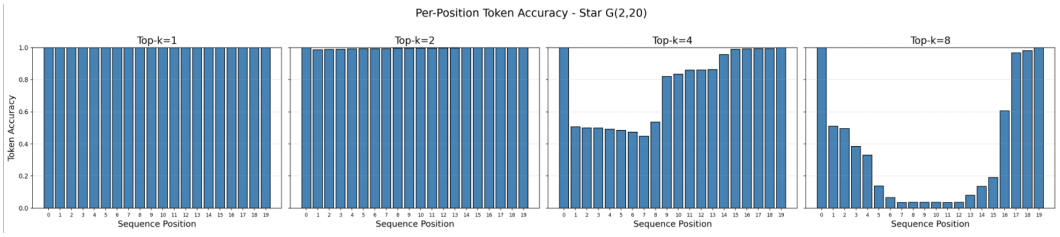

Figure 12: Decoding dynamics for $k = 4$ (left) and $k = 8$ (right) on the $G(2, 20)$ graph. As the number of simultaneously unmasked tokens increases, the strictly sequential reverse-decoding strategy breaks down, and the model prematurely attempts forward, higher-order predictions.

A.5.2 Block-wise decoding

The experiments above use a single generation block, allowing the model to condition on partially decoded tokens across the entire output sequence. In practical settings, however, outputs longer than the generation block must be produced block-wise, preventing later blocks from providing information while earlier blocks are generated. To examine how this constraint affects the learned strategy, we evaluate the NAR model on $G(2, 10)$ using two blocks of five path tokens each (Fig. 14). During generation of the first block, target-side information is unavailable, and the model therefore decodes the block in the forward direction. Once generation enters the second block and the target becomes available, the decoding order changes, the model anchors at the final path position and proceeds predominantly backward through the block. Thus, block-wise generation prevents the model from exploiting the global reverse-decoding shortcut across block boundaries, but the model

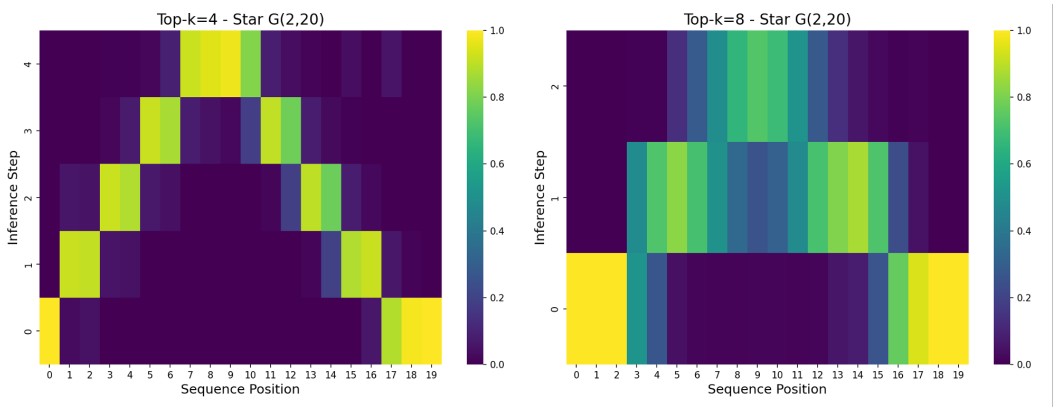

Figure 13: Mean per-position token accuracy across different top-k unmasking configurations on the $G(2, 20)$ test set.

still adaptively exploits deterministic reverse transitions whenever the required information becomes available.

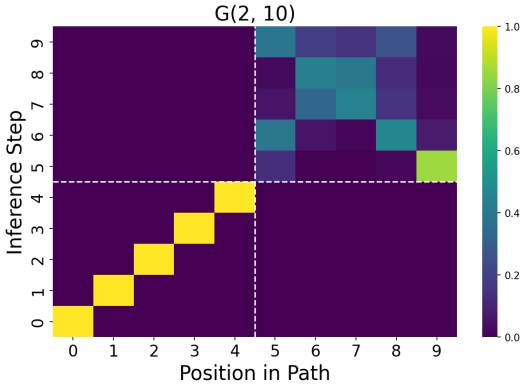

Figure 14: Block-wise decoding dynamics of the NAR model on $G(2, 10)$ using two generation blocks of five path tokens each. Dashed lines indicate the block boundary. Color intensity indicates the fraction of examples in which each path position is unmasked at each inference step. During the first block, target-side information is unavailable and the model decodes from left to right. Once the second block becomes active and the target is accessible, the model anchors at the final path position and decodes predominantly backward within the block.

A.6 Training Dynamics

We further examine the convergence of individual token positions (Fig. 15, Appendix A.6). While both paradigms quickly learn to copy the path endpoints, their mastery of intermediate vertices differs fundamentally. The AR model exhibits a distinct "bottleneck", struggling to predict the intermediate path before all middle vertices emerge simultaneously. Conversely, the NAR model learns incrementally, converging on correct predictions backward from the path's end. Freed from strict sequential dependencies, it capitalizes on local 1st-order transitions while the AR counterpart must globally resolve the overarching ℓ^{th}-order dependencies. By solving these easier sub-tasks and using converged vertices as a scratchpad for context, the NAR model implements an efficient alternative that bypasses the AR bottleneck entirely.

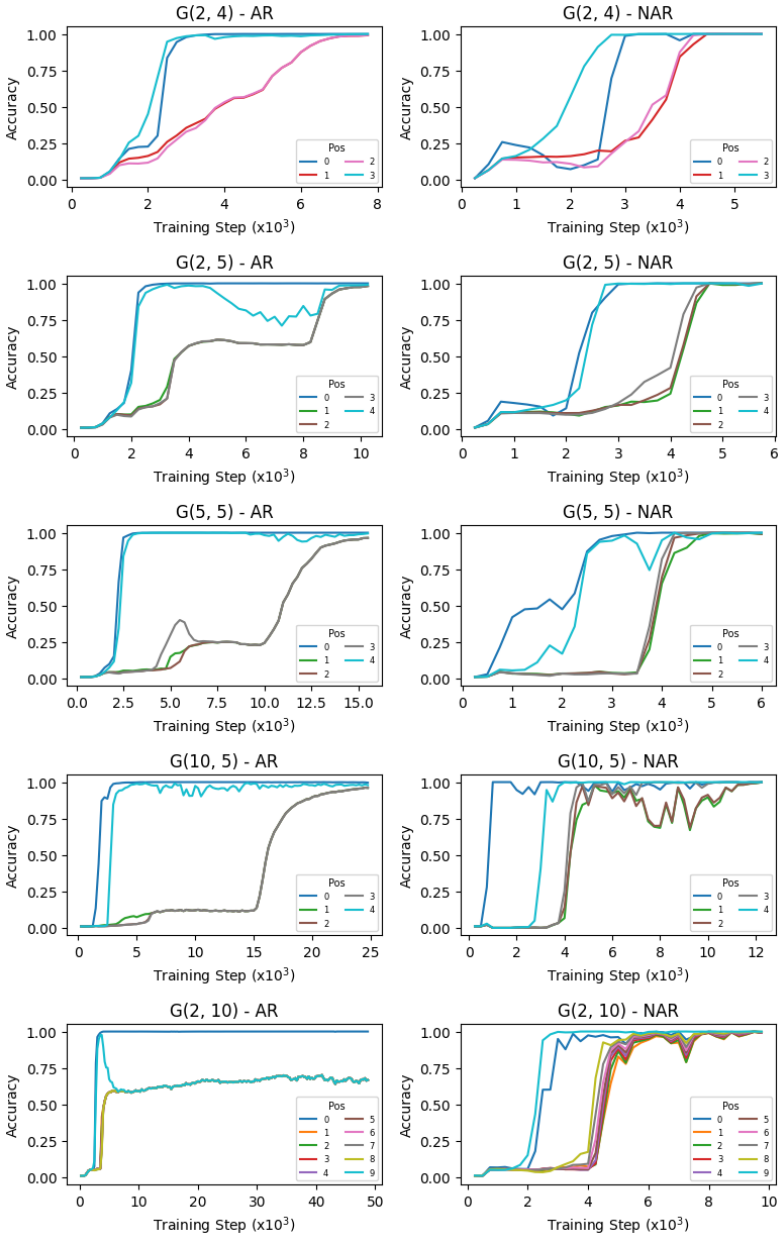

Figure 15: A comparison of the training dynamics of AR and NAR models depicting the per-position (vertex) average accuracy over the validation set. Note that "*G(2,10) - AR*" did not reach convergence. For visualization purposes, the x-axis scale varies across.

A.7 Robustness to Model Scale

The main experiments use a fixed Transformer model size, raising the possibility that the observed difference between AR and NAR training dynamics is specific to that capacity. To test this, we repeat the original Star-Path experiments across multiple model scales, while keeping the task and training setup otherwise unchanged. Figure 16 shows the exact-match trajectories for $G(2,5)$, $G(5,5)$, and $G(10,5)$ for both AR and NAR models. The original 21.5M-parameter runs are included as a reference.

Across model scales, the central qualitative behavior is preserved. Increasing model capacity generally reduces the number of training examples required to reach high accuracy, but it does not eliminate the difference between the two generative paradigms. NAR models consistently reach high exact-match accuracy substantially earlier in training, whereas AR models retain the characteristic intermediate regime before convergence. This distinction becomes particularly clear as graph complexity increases: on $G(5,5)$ and $G(10,5)$, increasing AR model capacity shifts the final transition to earlier in training, but does not remove the separation between the initial acquisition of simpler transitions and the later acquisition of the full lookahead solution.

These results further support our interpretation of the sample-complexity gap as a consequence of the different solution strategies available to AR and NAR models, rather than an artifact of insufficient model capacity. Additional capacity can make the higher-order computation easier to acquire, but it does not change the causal constraint faced by AR models: the correct forward transition must still be resolved before future path tokens are generated. In contrast, NAR models retain the ability to exploit bidirectional context and construct the solution from deterministic local transitions. Thus, model scale changes when the relevant behaviors are learned, but does not change the qualitative distinction in how the two paradigms learn to solve the task.

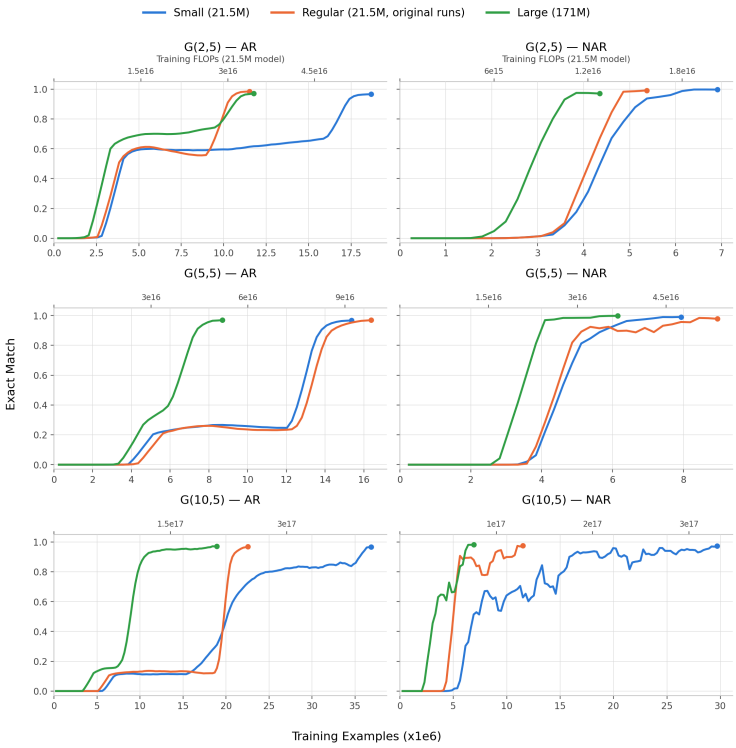

Figure 16: Robustness to model scale. Across smaller and larger model variants, NAR retains its sample-efficiency advantage over AR, while the characteristic difference in their learning dynamics remains qualitatively unchanged.

A.8 Comparing Alternative AR Architectures

To further isolate the source of the autoregressive behavior observed in the main experiments, we also evaluate a Mamba-based autoregressive model [18, 7, 21]. Since Mamba remains causally constrained, it cannot exploit the reverse-decoding strategy available to the NAR model, but it replaces the Transformer backbone with a recurrent state-space architecture. Thus, it allows us to separate effects that stem from autoregressive decoding itself from effects that may be specific to the Transformer implementation of AR modeling. We train the Mamba under the same data and optimization setup as the other models (See Table 2).

Figure 18 exhibits substantially smoother training dynamics for the Mamba architecture, suggesting that its internal computation of the graph structure differs from the one learned by Transformer self-attention. To probe this more directly, we measure the V-measure of representations with respect to topological distance from the center, following the same structural criterion used in §4.5. As shown in Fig. 17, Mamba maintains a substantially stronger distance-from-center signal across depth than NAR, and in many layers also stronger than Transformer AR. This suggests that Mamba constructs a more persistent representation of graph structure from the graph description alone.

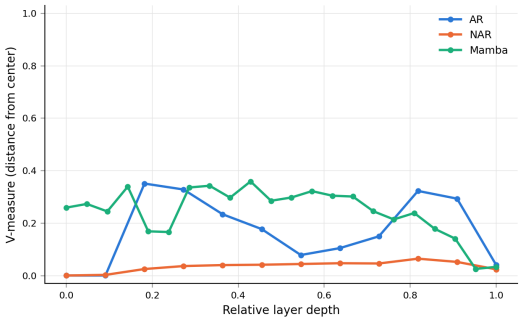

Figure 17: V-measure of latent representations with respect to topological distance from the center, plotted across relative layer depth for Transformer AR, NAR, and Mamba. Mamba maintains a stronger and more persistent structural signal than NAR, and in many layers stronger than Transformer AR, suggesting it encodes graph topology more effectively from the graph-description prefix.

At the same time, Mamba does not remove the core limitation of autoregressive decoding, as unlike NAR, it still cannot adapt generation order to the structure of the task. Thus, even if Mamba represents the graph more effectively than Transformer AR, it does not gain the reverse-decoding shortcut that underlies the NAR sample-efficiency advantage.

Overall, these findings refine our interpretation of the main results. The Mamba results suggests that alternative AR backbones may learn smoother and structurally richer internal representations than Transformer AR, although limited by causal decoding. A fuller mechanistic analysis of how Mamba stores and composes graph structure is left for future work.

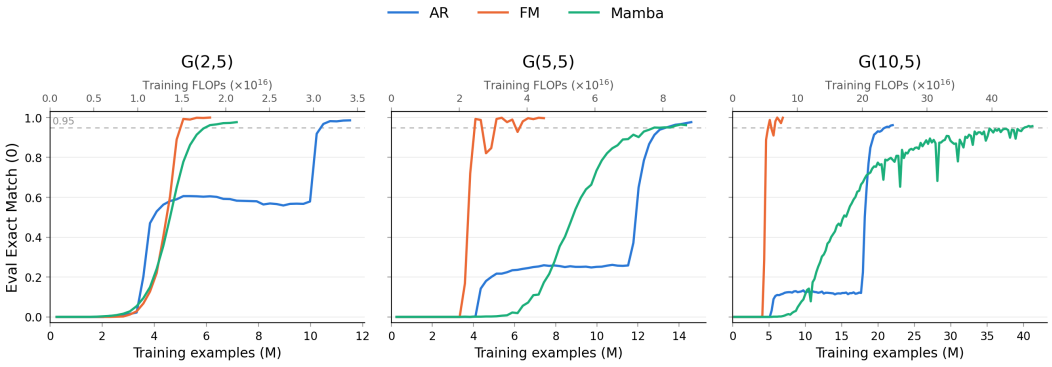

Figure 18: Training dynamics of Transformer AR, NAR (dLLMs), and Mamba AR models across increasingly complex Star-Path configurations. Mamba exhibits smoother learning than Transformer AR, but NAR retains a substantial sample-efficiency advantage as graph complexity increases.

Parameter	Mamba	Parameter	Mamba
Batch Size	1024	Causal Masking	✓
Max. Training Examples	50×10^6	Autoregressive Decoding	✓
Optimizer	AdamW	Loss Function	Cross Entropy
Learning Rate	3×10^{-4}	Source Distribution	Masked Seq.
β_1	0.9	Scheduler Type	Polynomial
β_2	0.95	Hidden Size	384
ϵ	1×10^{-8}	Cond. Dim.	$\times$
Weight Decay	0.03	# Mamba Blocks	22
Warmup Steps	5×10^3	State Size	16
Grad Clip	1.0	Expansion Factor	2
$\eta_{\min}$ Ratio	0.01	Conv. Kernel Size	4
Accumulation Steps	1	Dropout	0.0

Table 2: Hyperparameters for the Mamba baseline. Training hyperparameters are matched to the AR and NAR models in Table 1.

A.9 Generalization Beyond Star-Path

Our use of the synthetic Star-Path task coupled with dedicated training deliberately isolates the effect of the generative paradigm on lookahead and enables control over the cause of the effect. However, as such, it abstracts away some aspects of practical use cases. We therefore complement the mechanistic study with an external, more confounded, study of full practical AR and NAR LLMs using established planning benchmarks. We compare two diffusion language models (DiffusionGemma-26B-A4B-it [17, 16] and Dream-7B [38]), against three autoregressive (AR) baselines (Gemma-4-26B-A4B-it [16], Qwen2.5-7B [26], and Qwen3.5-9B-Base [27]). Particularly, DiffusionGemma is derived from Gemma-4, and Qwen2.5-7B is not instruction-tuned and does not have a dedicated post-trained thinking/reasoning mode. Importantly, Gemma-4 and Qwen3.5 are post-trained and therefore have direct access to scratchpad, alleviating the reverse decoding advantage of NAR models as discussed in 1.

A.9.1 Tasks

We evaluate on three benchmarks that require decisions whose validity depends on non-local future constraints.

Trip Planning [41] [4] is an itenerary planning dataset. Given cities to visit, required stay durations, and available direct flights, the model must produce a complete day-by-day itinerary. To solve the task efficiently, the model must therefore maintain a globally consistent trajectory rather than make independently correct local choices. We use all 1,600 test instances, spanning 3–10 cities.

Countdown [37], a model must combine the input numbers using $+, -, \times, \div$ to reach a target value. Each arithmetic choice changes the set of reachable future states, making successful solutions depend on multi-step lookahead rather than isolated next-token prediction. We use 2,976 test tasks.

Sudoku [37], the model is prompted to solve a given Sudoku table. We use 100 test tasks.

A.9.2 Experimental setup

All models are evaluated purely in context, without task-specific fine-tuning, following the few-shot protocol of Dream-7B [38]. Dream uses its published diffusion inference settings, with the number of denoising steps equal to the output-token budget: 256 for Trip Planning, 32 for Countdown, and 24 for Sudoku.

A.9.3 Results

Figure 19 shows an overall advantage for diffusion language models, with the clearest gains on Countdown and Sudoku. Diffusion models consistently outperform the autoregressive baselines, and DiffusionGemma is the strongest model evaluated. These results further show the advantage of NAR

[4] `https://github.com/google-deepmind/natural-plan`

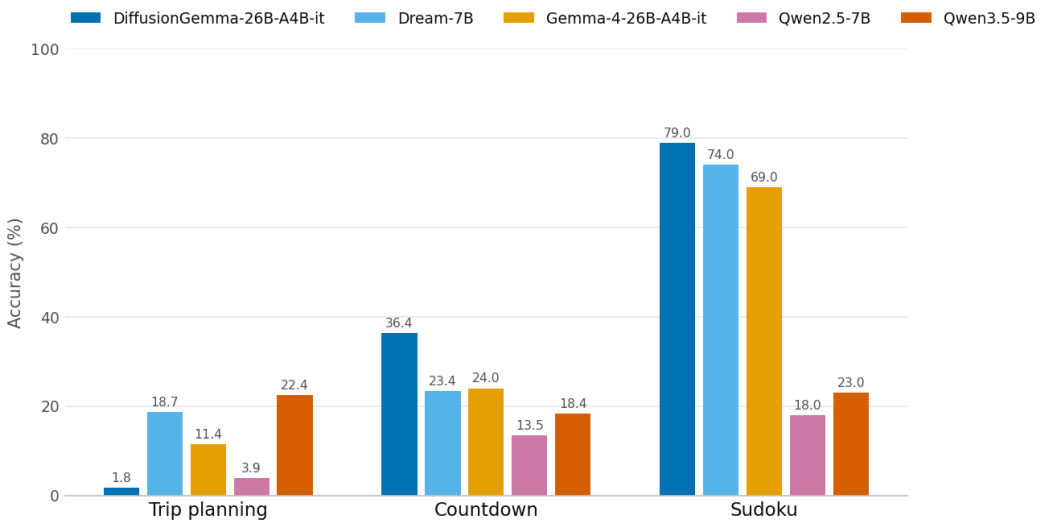

Figure 19: AR and NAR model performance on practical planning tasks using commonly used fully-trained LLMs.

models on lookahead tasks, and especially support the generalization of our hypothesis to practical setups.

The only exception is observed in the Trip-Planning task where Dream diffusion model remains competitive, but DiffusionGemma performs substantially worse than the other models. A closer analysis of Trip-Planning suggests that DiffusionGemma's Trip Planning score is strongly affected by its inability to produce an output compatible with the output-parser.

Taken together, these results provide complementary behavioral support for our controlled findings. The Star-Path experiments show how non-causal generation can exploit information across the output trajectory to avoid difficult forward commitments, while the full-scale evaluation shows that diffusion models are also strong on established planning benchmarks characterized by interdependent decisions and global constraints.

Since released checkpoints differ in model size, pretraining, post-training, and inference procedure and in particular, the Gemma variants are instruction-tuned, while Qwen2.5-7B is a base checkpoint without instruction tuning or a dedicated post-training thinking mode, we interpret these results as supporting the overall behavioral advantage of diffusion models, rather than as a causal comparison of the two generative paradigms.

A.10 Latent Space Implications - Additional Results

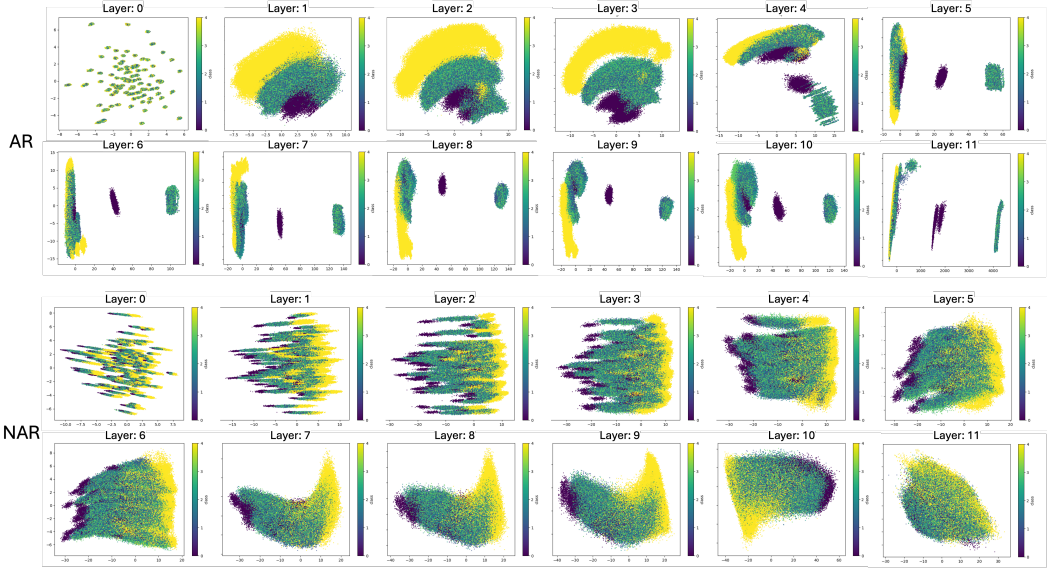

Figure 20: PCA projection of latent representations across layers for AR (top) and NAR (bottom) models on $G(5, 5)$. Points are colored by topological depth (Purple: Center, Yellow: Leaf). While AR maintains structural separation deep into the network, NAR's structural representation becomes diffuse in later layers.

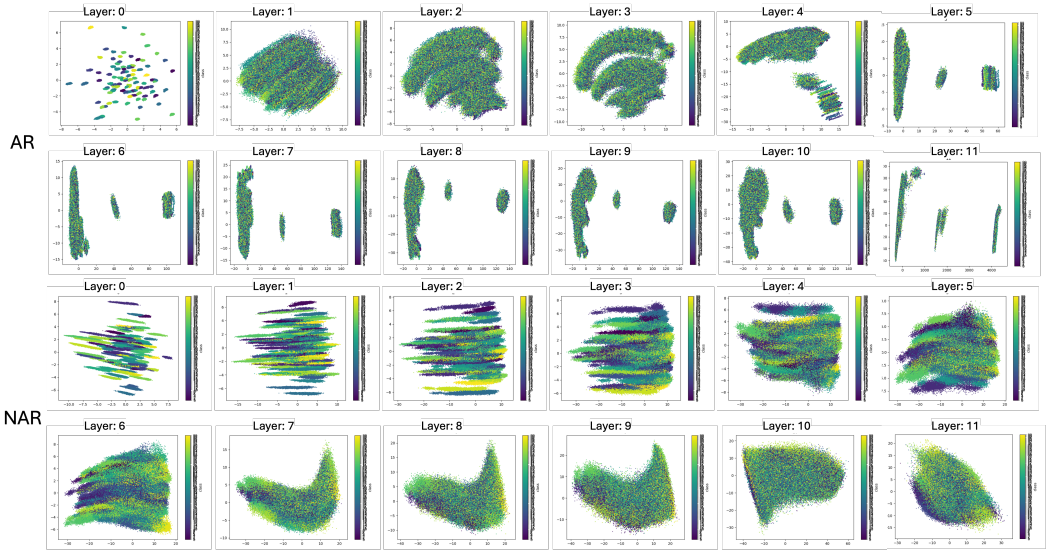

Figure 21: PCA projection of latent representations colored by vertex ID. NAR models (bottom) preserve identity-based clustering deeper into the network compared to AR models (top), suggesting a reliance on local identity matching.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive, or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in the appendix. If you answer [Yes] to a question, in the justification, please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers, and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the claims regarding the performance and mechanisms of non-autoregressive (NAR) versus autoregressive (AR) models on lookahead planning tasks, which are directly supported by the empirical results and mechanistic analyses detailed in Sections 4 and 5.

   Guidelines:

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

2. **Limitations**

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

Answer: [Yes]

Justification: We explicitly address the limitations of our work in a dedicated "Limitations" subsection at the end of the paper.

Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper does not include theoretical results.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

Justification: The core experimental setup is described in Section 3.3, and comprehensive details regarding the model architecture, baseline implementations, and data generation are provided in Appendix A.1 as well as the code.

Guidelines:

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

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

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

   Answer: [No]

   Justification: Code and data will be submitted with the camera-ready version.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We provide a detailed configuration table (Table 1) in Appendix A.1 that specifies all necessary hyperparameters, including batch sizes, learning rates, optimizer choices, and model dimensions.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: We reports statistical variance by including shaded error areas and confidence intervals in the primary plots, including Figures 2, 5, and 8, which represent the variance calculated across 10 distinct seeded runs.

   Guidelines:

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

8. **Experiments compute resources**

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

Answer: [Yes]

Justification: Appendix A.1 explicitly discloses the computational resources

Guidelines:

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

9. **Code of ethics**

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

Answer: [Yes]

Justification: The research involves the fundamental study of language model mechanisms using purely synthetic graph structures (Star-Path tasks) and conforms to standard ethical guidelines without involving human subjects, crowdsourcing, or harmful applications.

Guidelines:

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

10. **Broader impacts**

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

Answer: [Yes]

Justification: The paper explicitly addresses the societal impact of AI safety and alignment by highlighting a vulnerability in modern LLM training where models might exploit any available asymmetry or artifact instead of learning true causal reasoning.

Guidelines:

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

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We properly attribute the foundational code for their implementations by explicitly linking to and citing.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [Yes]

    Justification: The newly introduced assets, specifically the synthetic Star-Path and Two-Sided-Star task formulations, are thoroughly defined, mathematically formalized, and documented in Section 3.1 and Appendix A.4.

    Guidelines:

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

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

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

    Answer: [N/A]

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

    Guidelines:

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

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

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

    Answer: [N/A]

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

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

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

    Guidelines:

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