Improving Pathfinding with Anchoring Tokens

Huaqing Zhang [1] **Bingbin Liu** [2] **Juno Kim** [3] **Andrej Risteski** [4]

Abstract

Planning is a critical aspect of multi-step reasoning, yet it remains challenging for large language models (LLMs). In this work, we use pathfinding in graphs as a sandbox for understanding and improving the planning abilities of LLMs. Our results show that while conventional autoregressive training generalizes poorly, an *anchoring* strategy, whereby a model first predicts a small subset of intermediate nodes along the path, significantly improves the path finding performance. We confirm these gains on two families of graphs with markedly different structures and provide preliminary heuristics for selecting effective anchor nodes, offering guidance for more realistic settings.

1. Introduction

Planning is a critical aspect of multi-step reasoning, which has recently emerged as a central component towards unlocking many capabilities in large language models (LLMs). Many real-world applications, spanning mathematical puzzles or proofs, algorithmic tasks, program synthesis, strategic games, and agentic AI, appear to require complex reasoning chains, for which planning can be beneficial (Song et al., 2023; Valmeekam et al., 2023; Hirsch et al., 2024; Zhou et al., 2024; Stechly et al., 2025a;b). However, planning remains both challenging and poorly understood. Despite ongoing progress, many recent works benchmarking LLM planning capabilities still reveal various limitations (Kambhampati et al., 2024; Mirzadeh et al., 2024; Hirsch et al., 2024; Lin et al., 2024; Stechly et al., 2025b). Therefore, a deeper understanding of the principles underlying planning is essential to develop and deploy effective reasoning models.

To stress-test such capabilities, graph-based tasks—in particular *pathfinding*—serve as a conceptually clean yet versatile proxy for multi-step reasoning: many reasoning tasks reduce to navigating from a specified start state to a desired goal state over a graph of latent or explicit transitions. This has motivated mechanistic studies of algorithmic behavior in small-scale models (Khona et al., 2024; Abbe et al., 2024; Cohen et al., 2025) and studies probing reasoning of large-scale LLMs (Momennejad et al., 2023; Saparov & He, 2023; Dziri et al., 2023), as well as theoretical characterizations of architectures such as autoregressive Transformers (Sanford et al., 2024; Wang et al., 2024; Kim et al., 2025b). Further discussion of related work is provided in Appendix A.

In this work, we use **pathfinding in graphs** as a sandbox to quantitatively study the benefits of planning for multi-step reasoning problems. Given a source-target pair of connected nodes (S, T) in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, our goal is to produce a path from S to T. Standard autoregressive training is known to suffer from undesirable "shortcuts" (Wang et al., 2024; Bachmann & Nagarajan, 2024), which makes the model unable to generalize to unseen (S, T) pairs. We show that prepending **anchoring nodes** to these paths can be surprisngly effective at improving both training efficiency and generalization. Such anchoring nodes are selected from the path and can be considered as an abstraction of planning or subgoal-setting (Gandhi et al., 2025; Zeng et al., 2025). Although a concurrent study by Thankaraj et al. (2025) has demonstrated the empirical value of similar techniques, our goal is to unpack *why* anchoring works, and ultimately to understand *how* we should pick anchors in practical planning schemes. Concretely, using two distinct types of graphs (Section 2), we show:

- **Anchoring boosts pathfinding accuracy** (Section 3): We demonstrate performance gains on both types of graphs, confirming and extending the results of Thankaraj et al. (2025). For the K-partite graph, we provide a quantitative explanation on where the improvement comes from.

[1]Institute for Interdisciplinary Information Sciences, Tsinghua University [2] Kempner Institute, Harvard University [3]Department of Electrical Engineering and Computer Sciences, UC Berkeley [4]Machine Learning Department, Carnegie Mellon University. Correspondence to: Huaqing Zhang <zhanghq22@mails.tsinghua.edu.cn>, Andrej Risteski <aristesk@cs.cmu.edu>.

Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025, Vancouver, Canada.

- **Towards a principled choice of anchors** (Section 4): By analyzing the failure modes of planning-free training, we identify two heuristic criteria (*easiness* and *usefulness*) of an anchor node, which can be used to provide guidance for how to select anchor nodes.

Our anchoring formulation facilitates fine-grained analysis of rudimentary planning for long reasoning chains while maintaining low computational cost. The runs complete mostly within 1 hour and no more than 5 hours on a single GPU, allowing us to systematically vary hyperparameters that control problem complexity and gain insights into scalability.

2. Setup

Following recent work to probe the reasoning capabilities of language models (Wang et al., 2024; Khona et al., 2024; Cohen et al., 2025; Sanford et al., 2024; Abbe et al., 2024; Kim et al., 2025b), we formulate a reasoning task involving pathfinding on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. We define $\text{conn}(S, T) = 1$ if and only if there exists a path between nodes S and T in the graph. Let $\mathcal{T} \subseteq \{(S, T) \in \mathcal{V} \times \mathcal{V} \mid \text{conn}(S, T) = 1\}$ denote a subset of connected node pairs. For each given $(S, T) \in \mathcal{T}$, model is tasked with finding a valid path $\texttt{[PATH]}_{(S,T)} = [S, v_2, v_3, \cdots, v_{k-1}, T]$ from S to T. We partition $\mathcal{T}$ into two disjoint subsets: $\mathcal{T}_{\text{train}}$ for training and $\mathcal{T}_{\text{eval}}$ for evaluation. The data only contains shortest paths when there are multiple paths from S to T.

Pathfinding with anchoring nodes. A standard approach for pathfinding is to perform next-token prediction directly on the sequence of nodes in the path. We propose a modified procedure in which the model first predicts a small set of *anchoring nodes* $\texttt{[ANCHORS]} := [a_1, \cdots, a_s] \subset \texttt{[PATH]}_{(S,T)}$, which serve as intermediate waypoints to guide the pathfinding process. In other words, we compare pathfinding using two formats:

- Format 1 (Standard pathfinding) S, T, <EoI>, $\texttt{[PATH]}_{(S,T)}$, <EoS>.
- Format 2 (Pathfinding with anchoring tokens): S, T, <EoI>, [ANCHORS], <EoA>, $\texttt{[PATH]}_{(S,T)}$, <EoS>.

For either format, the model is trained and evaluated on next-token prediction on the sequence. A generated path is considered incorrect if it contains an invalid edge, i.e., nodes v_i, v_{i+1} for which $E_{v_i, v_{i+1}} \notin \mathcal{E}$.

Graphs We consider two structurally distinct graphs: the K-partite graph and the two-level graph. The K-partite graph offers a simple setting for quantitatively comparing different anchoring positions, while the two-level graph introduces a clustering structure that better reflects the organization of real-world knowledge graphs (Kim et al., 2025b). Concretely:

- The (directed) **K-partite graph** is a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the vertex set $\mathcal{V}$ is partitioned into disjoint sets $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K$, each containing N nodes. The subsets are arranged in order and connections are randomly generated between consecutive layers: for each $1 \leq k < K$ and for each pair of nodes $u \in \mathcal{V}_k$ and $v \in \mathcal{V}_{k+1}$, the directed edge from u to v is included independently with probability $p \in (0, 1)$. We consider paths from the first to the last layer, i.e. $\mathcal{T} = \{(S, T) \in \mathcal{V}_1 \times \mathcal{V}_K \mid \text{conn}(S, T) = 1\}$.
- The (undirected) **two-level graph** $\mathcal{G}$ consists of N_1 clusters $\mathcal{V}_\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{N_1}\}$, where each cluster $\mathcal{C}_i$ consists of N_2 nodes. There are two types of edges: *intra*-cluster and *inter*-cluster edges. The intra-cluster edges are generated as an Erdos-Renyi graph in each cluster $\mathcal{C}_i$ with probability p_2: each pair of nodes inside $\mathcal{C}_i$ is connected independently with probability p_2. The inter-cluster edges are drawn by first generating an Erdos-Renyi graph $(\mathcal{V}_\mathcal{C}, \mathcal{E}_\mathcal{C})$ on the set of clusters with probability p_1, then connecting two random nodes $u \in \mathcal{C}_i$ and $v \in \mathcal{C}_j$ for each pair of connected clusters $(\mathcal{C}_i, \mathcal{C}_j) \in \mathcal{E}_\mathcal{C}$. We denote the cluster which v belongs to by $\mathcal{C}(v)$. The model is trained and evaluated on *inter*-cluster source-target pairs $\mathcal{T} = \{(S, T) \in \mathcal{V} \times \mathcal{V} \mid \mathcal{C}(S) \neq \mathcal{C}(T)\}$.

Figure 8 provides illustrations of both graph structures. The training data generation process is detailed in Appendix C.1 and Appendix C.2.

3. Anchoring Helps with Pathfinding

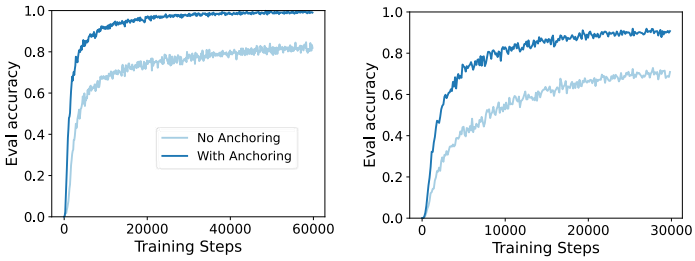
We show that anchoring improves pathfinding performance for both types of graphs; analyses are deferred to Section 4.

K-partite Graph We sample a random K-partite graph with $K = 9$, $N = 2500$, and edge probability $p = 0.001$. We use 10% of the source-target pairs in $\mathcal{T}$ for training and the remaining for evaluation. When trained with anchoring, the anchoring node is set to be the midpoint of a path, i.e. $v_{\lceil K/2 \rceil}$ in a path $\texttt{[PATH]}_{(S,T)} = [S, v_2, v_3, \cdots, v_{k-1}, T]$. This

choice is naturally determined from the graph structure: the model needs to learn all possible paths connecting $\mathcal{V}_1$ and $\mathcal{V}_K$; setting the anchoring nodes at the kth layer reduces the expected number of such paths from $\mathcal{O}(N(pN)^{K-1})$ to $\mathcal{O}\left(N(pN)^{k-1} + N(pN)^{K-k}\right)$, whose value is minimized with $k = \lceil K/2 \rceil$; see Appendix B.1 for more details. While this path-counting argument is specific to the K-partite graph, we will provide general heuristics for choosing anchoring nodes in Section 4.

As shown in Figure 1(a), anchoring significantly improves generalization performance.

Two-Level Graph In this experiment, we randomly sample a two-level graph with $N_1 = 50$, $N_2 = 100$, $p_1 = 1$, $p_2 = 0.2$, i.e., the clusters form a clique at the upper-level; we consider setups with $p_1 < 1$ in Section 4.2. The model is trained and evaluated on *inter*-cluster source-target pairs $\mathcal{T} = \{(S, T) \in \mathcal{V} \times \mathcal{V} \mid \mathcal{C}(S) \neq \mathcal{C}(T)\}$, of which 0.1% is used for training, and the rest is reserved for evaluation. The anchoring nodes are selected as the endpoints of the inter-cluster edge connecting $\mathcal{C}(S)$ and $\mathcal{C}(T)$, which we denote as (M_1, M_2). The path $\texttt{[PATH]}_{(S,T)}$ is composed of two intra-cluster paths $\texttt{[PATH]}_{(S,M_1)}$ and $\texttt{[PATH]}_{(M_2,T)}$, and we always use the shortest intra-cluster paths in training. More details are provided in Appendix C.2.



(a) K-partite graph (b) Two-level graph

Figure 1: Anchoring improves pathfinding on both the K-partite and two-level graphs.

As shown in Figure 1(b), anchoring significantly improves the evaluation accuracy on the two-level graph.

4. Understanding Anchoring

We wish to understand how anchoring facilitates pathfinding. We begin by distinguishing two error modes in a generated path. Building on these observations, we propose two heuristic criteria, *easiness* and *usefulness*, that capture how likely an anchoring node is to ameliorate each type of error and that empirically correlate with performance gain.

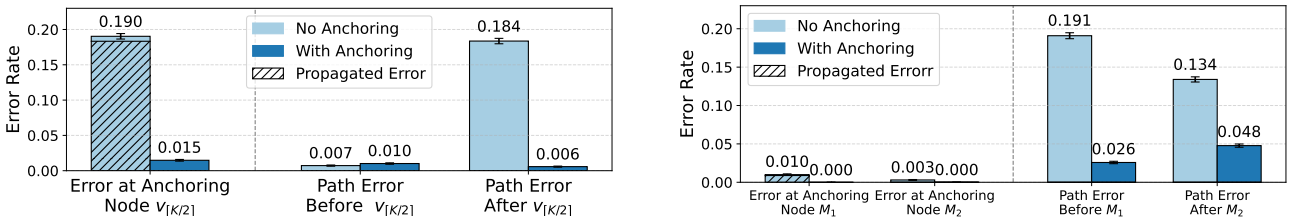
4.1. Failure Mode Analysis

We start with fine-grained analyses on the errors in pathfinding under both the K-partite graph and the two-level graph, where the anchoring strategies follow those previously described. Recall that a path is deemed erroneous if any of its edges is invalid. Based on this, we categorize errors into two types and observe that anchoring reduces both (Figure 2).

Errors at the Anchors. These are errors when predicting the anchor nodes. In the K-partite graph, an anchor $a \in \mathcal{V}_{\lceil K/2 \rceil}$ is considered incorrect if it is not connected to either the source S or the target T. In the two-level graph, anchors are the endpoints of the inter-cluster edge (M_1, M_2) of between $\mathcal{C}(S)$ and $\mathcal{C}(T)$. They are considered incorrect if they do not correspond to the actual inter-cluster edge. In the *No Anchoring* setup, the anchoring error is measured based on the nodes in $\texttt{[PATH]}_{(S,T)}$ that would serve as anchors in the corresponding *With Anchoring* setup (we refer to these nodes as *anchoring nodes*).

We observe that without anchoring, the majority of anchoring errors arise from error propagation: although the subpath leading up to the anchoring nodes is locally valid, it either fails to reach the target (as in the K-partite graph), or deviates from the training distribution due to length bias (as in the two-level graph), resulting in incorrect anchor predictions. Specifically, in the K-partite graph, the model may generate a locally valid subpath from S to some node $v' \in \mathcal{V}_{\lceil K/2 \rceil}$, but v' is not connected to T. And in the two-level graph, since training uses the shortest intra-cluster subpaths, the model may deviate by generating a longer, suboptimal path. Due to length bias, it tends to jump from $\mathcal{C}(S)$ to $\mathcal{C}(T)$ prematurely—before reaching the correct inter-cluster endpoint M_1. This highlights a key limitation of next-token prediction: the model greedily follows local connections and fails at long-range planning.

Errors in Other Parts. Anchoring nodes also help reduce errors in other parts of the path—both before and after the anchoring positions.



(a) K-partite Graph

(b) Two-level graph

Figure 2: Error analyses on K-partite (left) and two-level graph (right). Anchoring reduces both errors on the anchors and the subpaths before and after. For errors on anchors, a majority are due to previous nodes being off the shortest path.

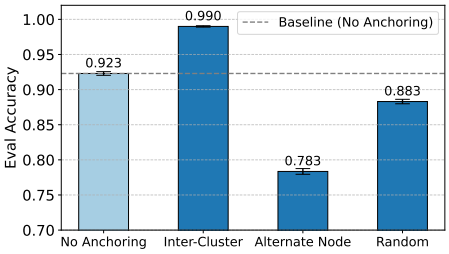


Figure 3: Comparing anchoring choices for two-level graphs. Anchors reflecting the graph structure (i.e. endpoints of cross-cluster edges) prove to be most effective.

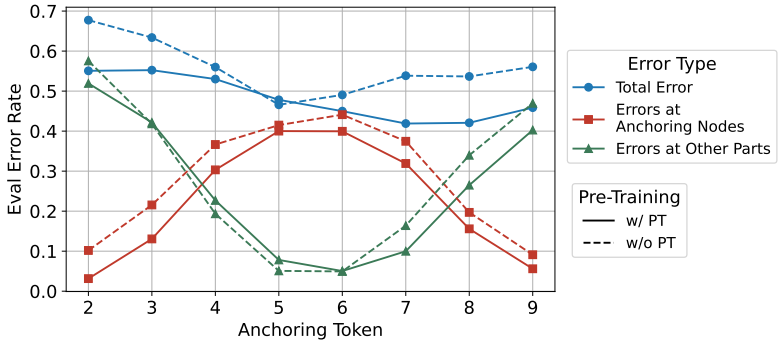


Figure 4: Comparing anchoring choices for K-partite graphs where $K = 10$. While anchors at layer $k = 5$ is the most effective based on the graph structure, pretraining (at layer $k = 7$) moves the optimal anchor choice.

Anchoring Reduces Both Types of Errors. By generating the anchors before the full path, the prediction of anchoring nodes depends only on the input (S, T), rather than on the (possibly erroneous) subpath leading up to them. This helps eliminate error propagation. To ensure this works well, we want the anchors to be easy to predict based solely on S and T. Indeed, in the two-level graph, the anchors (M_1, M_2) depend only on the high-level graph structure—that is, the inter-cluster edge between $\mathcal{C}(S)$ and $\mathcal{C}(T)$, which makes the easier to learn. In the K-partite graph, we provide a quantitative explanation of why anchor prediction is easier in Appendix B.2.

Once the anchors are correctly predicted, the remaining path generation can explicitly condition on them, which simplifies the task by splitting the full path into two shorter subpaths. The above discussion motivates our two heuristics for anchor selection: *easiness* and *usefulness*.

4.2. Choosing Anchors: Easiness and Usefulness

The error analyses above motivate two heuristics to guide the selection of anchoring nodes:

- *Easiness*: This corresponds to the error rate of the anchoring nodes; easier to predict nodes are preferable as anchors.
- *Usefulness*: This corresponds to the amount of error reduction on other nodes in a path.

The optimal anchoring node should balance the trade-off between *easiness* and *usefulness*. For example, in the K-partite graph, when the graph is relatively dense (i.e., $(pN)^{K-1} \gg N$), nodes in layers closer to S or T are easier to predict, as nearly any node in early layers can reach the target, and nearly any node in late layers can be reached from the source. In contrast, anchoring near the midpoint is more useful for guiding the rest of the path. In this case, *usefulness* turns out to be more important, as evidenced by Figure 4. See also Appendix B.1 for a more detailed discussion.

The exact tradeoff and strengths of these two factors are affected by both the graph structure and pretraining.

Effect of Graph Structure The graph structure affects the choice of anchors. In this section, we consider a two-level graph with $N_1 = 200, N_2 = 50, p_1 = 0.1, p_2 = 0.4$. Different from the setup in Section 3, the upper-level clusters now may not always be neighboring, but form an Erdos-Renyi graph with $p_1 = 0.1$. We compare different anchoring strategies below, where the anchoring nodes are set to be 1) endpoints of inter-cluster edges; 2) every other node in the path; or 3) random nodes sampled from the path. The number of anchoring nodes are kept approximately the same across all three strategies. As shown in Figure 3, anchoring at key structural points of the graph—particularly inter-cluster edges—is significantly more effective than uniformly or randomly placing anchors.

Effect of Pretraining LLM training usually consists of a generic pretraining phase before reasoning-specific training. We will show that pretraining interacts strongly with anchoring and substantially influences the optimal anchor choice. In the following, we first "pretrain" the model on vanilla source-target paths, and then continue to a "fine-tuning" stage with anchoring. The two stages use disjoint sets of source-target pairs $\mathcal{T}^{\text{stg1}}, \mathcal{T}^{\text{stg2}} \subset \mathcal{T}$ such that paths in $\mathcal{T}^{\text{stg2}}$ are compositions of paths in $\mathcal{T}^{\text{stg1}}$. This mimics compositional generalization, where pretraining provides broad foundational knowledge and fine-tuning learns more complex tasks by building on the knowledge from pretraining.

We experiment with the K-partite graph, for $K = 10, N = 1500$ and $p = 0.002$. We use all pairs in $\mathcal{T}^{\text{stg1}}$ for pretaining. At the "fine-tuning" stage, 0.5% of pairs in $\mathcal{T}^{\text{stg2}}$ are used for training and the remaining for evaluation. As discussed earlier, when there is no pretraining, the midpoint is the optimal choice for trading off easiness and usefulness. Pretraining can change such a tradeoff. Specifically, we pretrain the model on source-target pairs for which $S \in \mathcal{V}_1, T \in \mathcal{V}_k$ or $S \in \mathcal{V}_k, T \in \mathcal{V}_K$; we set $k = 7$ in the experiments. As shown in Figure 4, such pretraining shifts the optimal anchor choice from layer 5 to layer 7, as pretraining makes anchoring at layer 7 more useful. At a high level, this points to a possible strategy for anchor selection: placing anchors at the interface between different pieces of knowledge learned during pretraining. Such junctions may serve as natural points for composition, enabling the model to better combine what it has learned in pretraining to solve more complex tasks.

5. Conclusion and Discussion

This work demonstrates that anchoring tokens, a tractable analogy to planning, provide a simple yet powerful mechanism to improve pathfinding performance. By systematically analyzing error modes, we identify a key limitation of next-token prediction: the model greedily follows local connections but struggles with long-range planning. We show that anchoring can effectively mitigate this issue. Furthermore, we uncover practical heuristics—*easiness* and *usefulness*—that explain when and why anchoring helps. Our results also reveal how graph structure and pretraining interact with anchoring, influencing the optimal anchor selection strategy.

Our work provides an initial step toward understanding anchoring strategies in sequence prediction. While we show that even simple anchoring can effectively reduce error propagation and improve pathfinding, many important questions remain open. A key direction for future work is to establish theoretical guarantees for when and why anchoring helps. Another promising avenue is to move beyond simple anchoring based on fixed token positions, and explore more flexible strategies such as hierarchical anchoring. Although we propose two practical heuristics—*easiness* and *usefulness*—to guide anchor selection, it remains unclear how to choose anchors in more general settings. One possible approach is to train the model to place anchors automatically using reinforcement learning. Interestingly, behavior similar to anchoring has been observed in reasoning models trained with reinforcement learning, where subgoal setting emerges as a characteristic pattern of complex multi-step reasoning (Gandhi et al., 2025; Zeng et al., 2025).

References

Abbe, E., Bengio, S., Lotfi, A., Sandon, C., and Saremi, O. How far can transformers reason? the globality barrier and inductive scratchpad. In *Advances in Neural Information Processing Systems*, 2024.

Arriola, M., Gokaslan, A., Chiu, J. T., Yang, Z., Qi, Z., Han, J., Sahoo, S., and Kuleshov, V. Block diffusion: Interpolating between autoregressive and diffusion language models. *International Conference on Learning Representations*, 2025. doi: 10.48550/arXiv.2503.09573.

Bachmann, G. and Nagarajan, V. The pitfalls of next-token prediction. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=76zq8Wkl6Z.

Bavarian, M., Jun, H., Tezak, N., Schulman, J., McLeavey, C., Tworek, J., and Chen, M. Efficient training of language models to fill in the middle. *arXiv preprint arXiv: 2207.14255*, 2022.

Berglund, L., Tong, M., Kaufmann, M., Balesni, M., Stickland, A. C., Korbak, T., and Evans, O. The reversal curse: Llms trained on "a is b" fail to learn "b is a". *International Conference on Learning Representations*, 2023.

Chen, Z., White, M., Mooney, R., Payani, A., Su, Y., and Sun, H. When is tree search useful for llm planning? it depends on the discriminator, 2024. URL https://arxiv.org/abs/2402.10890.

Cohen, A., Gromov, A., Yang, K., and Tian, Y. Spectral journey: How transformers predict the shortest path, 2025. URL https://arxiv.org/abs/2502.08794.

Dong, L., Yang, N., Wang, W., Wei, F., Liu, X., Wang, Y., Gao, J., Zhou, M., and Hon, H. Unified language model pre-training for natural language understanding and generation. *Neural Information Processing Systems*, 2019.

Dziri, N., Lu, X., Sclar, M., Li, X. L., Jiang, L., Lin, B. Y., West, P., Bhagavatula, C., Bras, R. L., Hwang, J. D., Sanyal, S., Welleck, S., Ren, X., Ettinger, A., Harchaoui, Z., and Choi, Y. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, 2023.

Gandhi, K., Chakravarthy, A., Singh, A., Lile, N., and Goodman, N. D. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv: 2503.01307*, 2025.

Hirsch, E., Uziel, G., and Anaby-Tavor, A. What's the plan? evaluating and developing planning-aware techniques for language models, 2024. URL https://arxiv.org/abs/2402.11489.

Hu, M., Mu, Y., Yu, X., Ding, M., Wu, S., Shao, W., Chen, Q., Wang, B., Qiao, Y., and Luo, P. Tree-planner: Efficient close-loop task planning with large language models, 2024.

Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L., and Murthy, A. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *International Conference on Machine Learning*, 2024.

Khona, M., Okawa, M., Hula, J., Ramesh, R., Nishi, K., Dick, R., Lubana, E. S., and Tanaka, H. Towards an understanding of stepwise inference in transformers: A synthetic graph navigation model, 2024. URL https://arxiv.org/abs/2402.07757.

Kim, J., Shah, K., Kontonis, V., Kakade, S., and Chen, S. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv: 2502.06768*, 2025a.

Kim, J., Wu, D., Lee, J., and Suzuki, T. Metastable dynamics of chain-of-thought reasoning: Provable benefits of search, rl and distillation. *International Conference on Machine Learning*, 2025b.

Li, X. L., Thickstun, J., Gulrajani, I., Liang, P., and Hashimoto, T. Diffusion-lm improves controllable text generation. *Neural Information Processing Systems*, 2022. doi: 10.48550/arXiv.2205.14217.

Lin, C.-C., Jaech, A., Li, X., Gormley, M., and Eisner, J. Limitations of autoregressive models and their alternatives. *North American Chapter of the Association for Computational Linguistics*, 2020. doi: 10.18653/V1/2021.NAACL-MAIN.405.

Lin, F., Malfa, E. L., Hofmann, V., Yang, E. M., Cohn, A., and Pierrehumbert, J. B. Graph-enhanced large language models in asynchronous plan reasoning, 2024. URL https://arxiv.org/abs/2402.02805.

Mirzadeh, S. I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., and Farajtabar, M. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. In *International Conference on Learning Representations*, 2024.

Momennejad, I., Hasanbeig, H., Vieira, F., Sharma, H., Ness, R. O., Jojic, N., Palangi, H., and Larson, J. Evaluating cognitive maps and planning in large language models with cogeval. In *Advances in Neural Information Processing Systems*, 2023.

Nagarajan, V., Wu, C. H., Ding, C., and Raghunathan, A. Roll the dice & look before you leap: Going beyond the creative limits of next-token prediction. *arXiv preprint arXiv: 2504.15266*, 2025.

Pannatier, A., Courdier, E., and Fleuret, F. σ-gpts: A new approach to autoregressive models. In Bifet, A., Davis, J., Krilavicius, T., Kull, M., Ntoutsi, E., and Zliobaite, I. (eds.), *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2024, Vilnius, Lithuania, September 9-13, 2024, Proceedings, Part VII*, volume 14947 of *Lecture Notes in Computer Science*, pp. 143–159. Springer, 2024. doi: 10.1007/978-3-031-70368-3_9. URL https://doi.org/10.1007/978-3-031-70368-3_9.

Peng, F. Z., Bezemek, Z., Patel, S., Rector-Brooks, J., Yao, S., Tong, A., and Chatterjee, P. Path planning for masked diffusion model sampling. *arXiv preprint arXiv: 2502.03540*, 2025.

Sanford, C., Fatemi, B., Hall, E., Tsitsulin, A., Kazemi, M., Halcrow, J., Perozzi, B., and Mirrokni, V. Understanding transformer reasoning capabilities via graph algorithms. In *Advances in Neural Information Processing Systems*, 2024.

Saparov, A. and He, H. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *International Conference on Learning Representations*, 2023.

Song, C. H., Wu, J., Washington, C., Sadler, B. M., Chao, W.-L., and Su, Y. Llm-planner: Few-shot grounded planning for embodied agents with large language models, 2023. URL https://arxiv.org/abs/2212.04088.

Stechly, K., Valmeekam, K., and Kambhampati, S. Chain of Thoughtlessness? An Analysis of CoT in Planning, 2025a. URL https://arxiv.org/abs/2405.04776.

Stechly, K., Valmeekam, K., and Kambhampati, S. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations*, 2025b.

Thankaraj, A., Jiang, Y., Kolter, J. Z., and Bisk, Y. Looking beyond the next token. *arXiv preprint arXiv: 2504.11336*, 2025.

Valmeekam, K., Sreedharan, S., Marquez, M., Olmo, A., and Kambhampati, S. On the planning abilities of large language models (a critical investigation with a proposed benchmark). In *Advances in Neural Information Processing Systems*, 2023.

Wang, S., Shen, Y., Feng, S., Sun, H., Teng, S.-H., and Chen, W. Alpine: Unveiling the planning capability of autoregressive learning in language models. In *Advances in Neural Information Processing Systems*, 2024.

Wei, H., Zhang, Z., He, S., Xia, T., Pan, S., and Liu, F. PlanGenLLMs: A Modern Survey of LLM Planning Capabilities, 2025. URL https://arxiv.org/abs/2502.11221.

Wong, L., Mao, J., Sharma, P., Siegel, Z. S., Feng, J., Korneev, N., Tenenbaum, J. B., and Andreas, J. Learning adaptive planning representations with natural language guidance, 2023. URL https://arxiv.org/abs/2312.08566.

Wu, X., Shen, Y., Shan, C., Song, K., Wang, S., Zhang, B., Feng, J., Cheng, H., Chen, W., Xiong, Y., and Li, D. Can graph learning improve planning in llm-based agents? In *Advances in Neural Information Processing Systems*, 2024.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

Zeng, W., Huang, Y., Liu, Q., Liu, W., He, K., Ma, Z., and He, J. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language agent tree search unifies reasoning acting and planning in language models. In *International Conference on Machine Learning*, 2024.

Zhu, H., Huang, B., Zhang, S., Jordan, M., Jiao, J., Tian, Y., and Russell, S. Towards a theoretical understanding of the 'reversal curse' via training dynamics, 2024. URL https://arxiv.org/abs/2405.04669.

Appendix

In the main paper, we showed that anchoring is an effective way to improve graph pathfinding performance and provided two heuristics for selecting anchors that can be informative for more realistic settings. In the appendix, we discuss related work and experimental details.

A. Related Work

Planning. The planning ability of LLMs have been evaluated on close-loop task planning (Hu et al., 2024), asynchronous planning tasks (Lin et al., 2024), algorithmic tasks (Stechly et al., 2025a), automated planning and math puzzles (Stechly et al., 2025b), and embodied agents (Song et al., 2023). Some works have argued that the planning ability of LLMs is still severely limited (Valmeekam et al., 2023; Hirsch et al., 2024; Lin et al., 2024; Chen et al., 2024), and advocated for the use of heuristic planners (Wong et al., 2023; Hirsch et al., 2024) or external verification (Kambhampati et al., 2024; Stechly et al., 2025b) to aid planning. Other works have developed LLM-based frameworks to improve their planning ability (Zhou et al., 2024; Lin et al., 2024; Wu et al., 2024). See also the survey by Wei et al. (2025) for more details on planning of LLMs.

Pathfinding and reasoning. Graph pathfinding has been used both as a sandbox to uncover reasoning mechanisms in small-scale transformer models, and to generate natural language reasoning tasks to probe the capabilities of large-scale LLMs. Khona et al. (2024) studies stepwise inference via the pathfinding (i.e., navigation) task in DAGs and reveals mechanisms such as path stitching and short path bias. Cohen et al. (2025) provides a mechanistic explanation of how two-layer Transformers find shortest paths in small graphs in-context, showing that the model learns to execute a novel spectral algorithm. Mostly similar to our setups, Wang et al. (2024) models planning as pathfinding in graphs and shows that Transformers solve this task by storing the adjacency and reachability matrices in model weights.

From a theoretical perspective, Sanford et al. (2024) studies the reasoning abilities of Transformers by characterizing their representational power to execute various graph algorithms, while Abbe et al. (2024) shows a learning-theoretic limit of Transformers on certain graph traversal tasks. In addition, Kim et al. (2025b) explicitly models chain-of-thought reasoning as a Markov process on an abstract graph to quantify the benefits of test-time compute.

For larger-scale models on more realistic setups, Momennejad et al. (2023) develops a planning ability evaluation protocol for LLMs based on synthetic graph navigation tasks (cognitive maps). Saparov & He (2023) also models first-order reasoning as linear DAGs and utilizes this connection to develop a synthetic QA dataset. Dziri et al. (2023) studies the performance of LLMs with scratchpads on compositional tasks such as multiplication and logic puzzles by formulating them as computation graphs and quantifying their level of complexity.

Beyond Next-Token Prediction The motivation for anchoring arises from fundamental limitations of next-token prediction (NTP). First, not all tokens in a sequence are equally easy to predict, so training can be dominated by a small set of tokens and critical but sparse signals can be overlooked (Bachmann & Nagarajan, 2024). Moreover, the strictly sequential nature of NTP imposes biases on the sequence order that can be hard to learn from (Lin et al., 2020), often resulting in unexpected failures such as reversal errors (Dziri et al., 2023; Berglund et al., 2023; Zhu et al., 2024). In response, several works have proposed variants of NTP that alter the prediction order (Yang et al., 2019; Dong et al., 2019; Bavarian et al., 2022), including predicting subsequences early which is closely related to anchoring (Thankaraj et al., 2025).

These observations naturally lead to a broader question: how about forfeiting autoregressive modeling altogether in favor of alternative paradigms? One such paradigm is *any-order prediction*, which encompasses discrete diffusion (Li et al., 2022; Kim et al., 2025a; Peng et al., 2025) as well as autoregressive models that determine their token ordering on the fly (Pannatier et al., 2024; Nagarajan et al., 2025). Hybrid schemes integrating both paradigms offer a particularly promising direction for future work (Arriola et al., 2025).

B. Additional Results

B.1. Why Anchoring Helps in K-Partite Path Finding: A Path-Counting Argument

In this section, we provide a quantitative explanation of why anchoring improves path finding in K-partite graphs. Given a source-target pair (S, T), predicting the next node $v_2 \in \mathcal{V}_2$ without anchoring requires the model to identify a node such that $\text{conn}(S, v_2) = 1$ and $\text{conn}(v_2, T) = 1$. The expected number of nodes satisfying the first condition is pN, which is small under our setup and therefore relatively easy to learn. However, the expected number of nodes satisfying the second

condition is approximately $(pN)^{K-2}$ (assuming $N > (pN)^{K-1}$), due to the exponential branching toward the target. By contrast, if we anchor at layer $k = \lceil K/2 \rceil$, the total number of candidates that need to be memorized in order to predict the anchor is minimized and balanced, with complexity $\mathcal{O}((pN)^{k-1} + (pN)^{K-k}) = \mathcal{O}((pN)^{\lfloor K/2 \rfloor})$. After generating the anchor, the remaining path can be decomposed into two shorter subpaths, which are easier to learn. This reasoning aligns with the empirical observations shown in Figure 5: Anchoring at the middle token achieves the least anchoring error, and simplifies full path generation.

Why does the anchoring node error in Figure 4 exhibit an inverted U-shape, seemingly inconsistent with Figure 5? There are two main reasons for this discrepancy:

First, in the setup of Figure 4 ($K = 10$, $N = 1500$, $p = 0.002$), we have $N < (pN)^{K-1}$. Under this regime, for a given target node T, almost any node in the early layers can connect to it, and similarly for nodes in the final layers. This makes prediction easier near the boundaries. However, nodes that are not close to the source or target layers remain difficult to predict.

Second, the training data is extremely limited in the setup of Figure 4—only 0.5% of all possible (S, T) pairs are used—making it difficult for the model to learn accurate connectivity patterns to and from the middle layers. As shown in Figure 6, when we increase the training data to 1% of all (S, T) pairs, the anchoring error curve becomes bimodal: tokens in the early and late layers are easy to predict because the model does not need to reason about long-range connectivity, while tokens in the middle layers become easier to predict as well, since the number of paths that must be memorized is minimized and more balanced. This supports the explanation provided above.

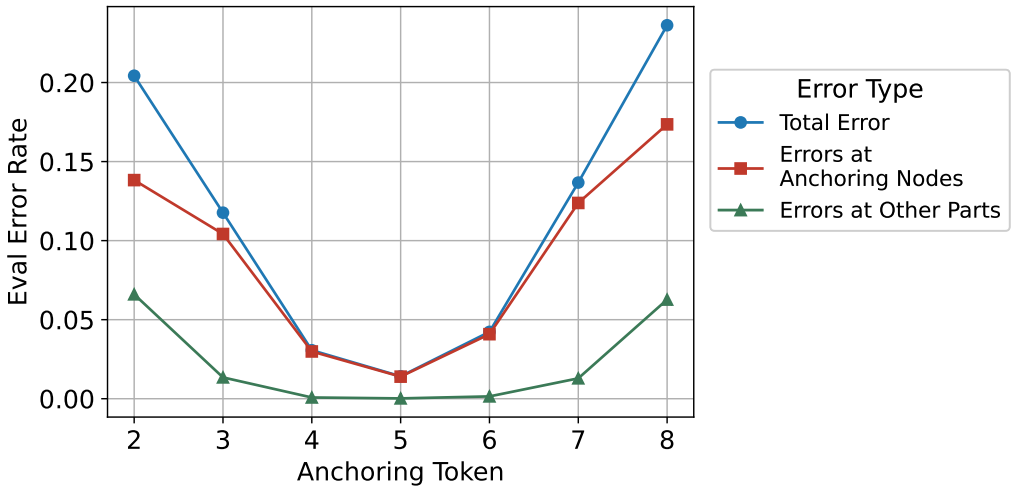


Figure 5: Anchoring with Different Token under the same Setup as Figure 1(b).

B.2. Pretraining for Two-level Path-Finding

In this section, we detail the pretraining process for the two-level graph described in Section 3 ($N_1 = 50, N_2 = 100, p_1 = 1, p_2 = 0.2$) and its effect. The pretraining process trains the model on all *intra*-cluster paths until near-perfect intra-cluster prediction accuracy is reached. Appendix B.2 shows the model's performance at the following fine-tuning stage (inter-cluster path-finding). Although all runs achieve near-perfect training accuracy, the inclusion of a pretraining stage improves the model's generalization. This effect is further amplified when anchoring is applied.

C. Experiment details

C.1. K-Partite Graph

To generate training data, a source-target pair is randomly sampled from $\mathcal{T}_{\text{train}}$, followed by a uniformly sampled path from S to T. For Figure 4, each model is pretrained for 100,000 steps and then fine-tuned for 30,000 steps. The optimizer state

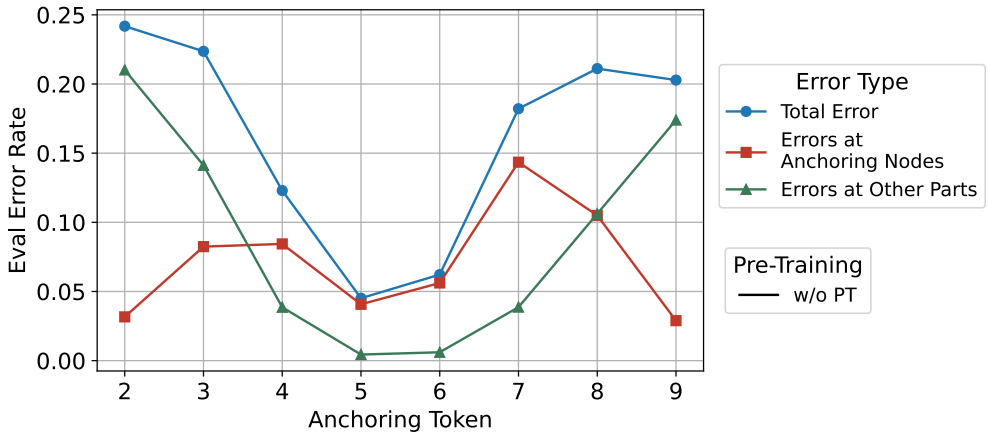


Figure 6: Anchoring with Different Token without Pretraining with 1% source-target pairs for training.

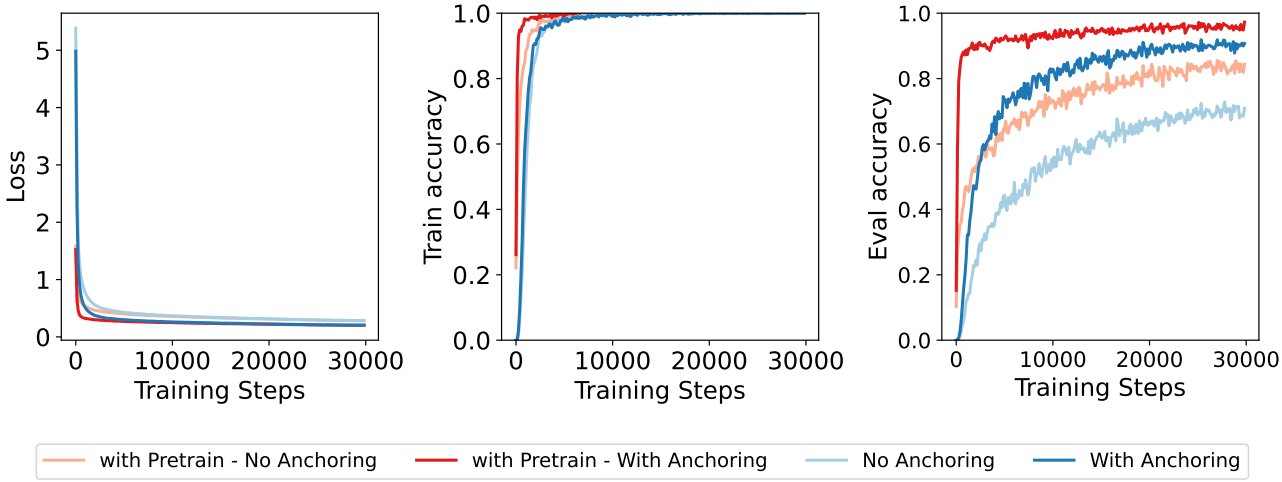


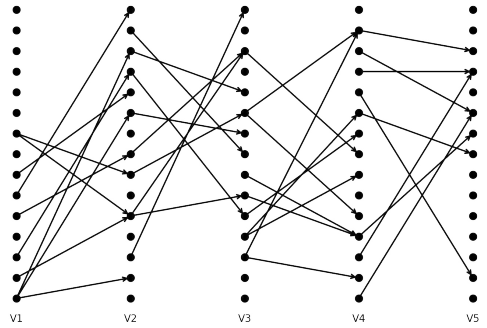
Figure 7: Pretraining improves the model's generalization in data-scarce settings. The addition of anchoring further enhances performance.

and learning rate scheduler are reset between pretraining and fine-tuning. For both Figure 2a and Figure 4, the error rate is evaluated on 10,240 samples.

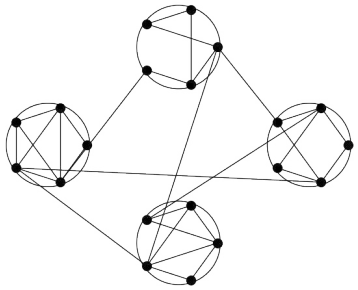
C.2. Two-Level Graph

To generate training data, a source-target pair (S, T) is randomly sampled from $\mathcal{T}_{\text{train}}$. A random shortest path is then sampled in the upper-level graph of clusters $(\mathcal{V}_{\mathcal{C}}, \mathcal{E}_{\mathcal{C}})$, connecting $\mathcal{C}(S)$ to $\mathcal{C}(T)$. Denote this cluster-level path by $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$, where $\mathcal{C}_1 = \mathcal{C}(S)$ and $\mathcal{C}_k = \mathcal{C}(T)$. For each consecutive cluster pair $(\mathcal{C}_i, \mathcal{C}_{i+1})$, let (u_i, v_i) be the edge connecting them, and define $v_0 = S$, $u_k = T$. The full path is constructed by concatenating the shortest intra-cluster sub-paths from v_i to u_i within each $\mathcal{C}_i$.

For Figure 3, the model is trained for 30,000 steps under each anchoring strategy. For both Figure 2b and Figure 3, model performance is evaluated on 10,240 samples. For Appendix B.2, the pretraining stage lasts 100,000 steps, followed by 30,000 steps of fine-tuning.



(a) A K-partite graph with $K = 5$, $N = 15$, and $p = 0.05$.



(b) A two-level graph with $N_1 = 4$, $N_2 = 5$, $p_1 = 1$, and $p_2 = 0.5$.

Figure 8: Examples of K-partite graphs and two-level graphs.

C.3. Additional Details

Hyperparameters. All models use a GPT-2 architecture with 3 layers and 3 attention heads, with a hidden dimension of 768. We train using the AdamW optimizer with a learning rate of 3×10^{-4} and no weight decay. A linear learning rate decay schedule is applied. The batch size is set to 256.

Computational Resources. Experiments were conducted on NVIDIA 3090 and A100 GPUs. Each run typically completes within 1 hour on a single GPU. The pretraining stage described in Section 4.2 takes up to 5 hours on a single GPU.