Stepwise Inference in Transformers: Exploring a Synthetic Graph Navigation Task

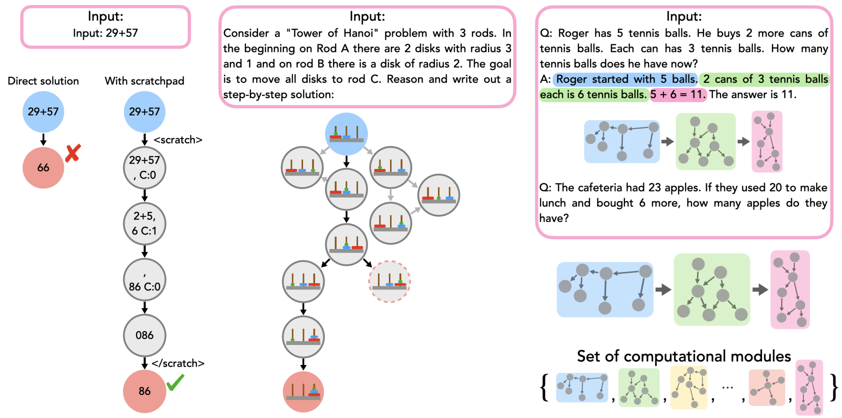
Anonymous Author(s)
Affiliation
Address
email

Abstract

Taking correct steps through elementary logical operations is the essence of logical reasoning, culminating in precise planning outcomes. While such *stepwise inference* approaches have demonstrated benefits in Large Language Models (LLMs), conducting an accurate quantitative evaluation is challenging, given their extensive scale, complexity, and lack of accessibility. We introduce a minimal synthetic setup, where an autoregressive language model solves a navigation task on directed acyclic graphs (DAGs), taking inspiration from computational graphs and execution traces. By implementing training with sample paths from start to goal node in a 'step-by-step' manner, we perform systematic experiments and develop novel analyses illustrating that stepwise navigation proves advantageous when the underlying graph is hierarchical and generalization necessitates the stitching of subpaths observed during pretraining. Further, we observe a diversity-accuracy tradeoff while varying sampling temperature and a bias towards generating shorter paths. We next elucidate how in-context chain-of-thought exemplars can steer the model's navigation. Importantly, these exemplars can guide the model to follow a path of reasoning we provide, instead of relying on its potentially biased priors. Together, this work showcases the utility and adaptability of this paradigm in exploring the complexities of logical reasoning and planning in LLMs.

1 Introduction

Here, we strive to formulate a framework that is not only simple, controllable, and interpretable but also encapsulates the essential features of an array of stepwise inference tasks such as scratchpad and zero/few-shot chain-of-thought in transformers. Our design philosophy adheres to the principle of constructing a model task that is *"as simple as possible, but not simpler"*, ensuring that the model embodies the following set of properties:: (i) the task can be better solved by outputting the intermediate steps of computation; (ii) there can be several possible paths of computational steps to solve the task; and (iii) the context of the task can be controlled by providing exemplars in the prompt. Here we argue that the paradigm of graph navigation problems provides such a fundamental framework. Inspired by computational graphs and execution traces, we model stepwise inference as navigating simple paths in a directed acyclic graph (DAG), representing a chain of logic (Dziri et al., 2023). Given a start and goal node, the transformer must autoregressively produce a sequence of nodes that concludes at the goal node. This task requires two levels of computation: locally, each step taken by the model must be valid, and on a global scale, the sequence of steps must be strategically planned in advance to reach the goal node. This setup enables us to modify (1) the structure of the underlying graph (2) the content of the training samples during pre-training and (3) the information provided to the model in-context before cue the model with the goal and start nodes. Consequently, we can systematically examine the impact of these properties on the development of reasoning abilities, an investigation that is challenging to conduct on a large scale. While (1)



(a) Scratchpad (b) Zero-shot chain of thought (c) Few-shot chain of thought

Figure 1: **Graph navigation task as a simple, steerable, and interpretable framework for exploring stepwise inference.** (a) Scratchpad (Nye et al., 2021) improves LLMs' ability to perform complext multi-step computations, such as arithmetic, when they write intermediate computation steps to a buffer called a scratchpad. (b) Zero-shot chain-of-thought prompting (Kojima et al., 2022) improes LLMs' ability to perform multi-step reasoning, such as Tower of Hanoi by prompting them to generate detailed reasoning paths. (c) Few-shot chain-of-thought prompting (Wei et al., 2022) improves LLMs' ability to perform multi-step reasoning, such as solving math word problems (Cobbe et al., 2021), by first presenting an exemplar in-context in the prompt.

and (2) together allows us to explore scratchpad and zero-shot step-by-step reasoning, which is relying on the model's internalized abilities, (3) also delves into few-shot in-context chain of thought prompting (Wei et al., 2022), where predictions are made with guiding examples. Specifically, we examine how in-context exemplars affect the path produced by the model and systematically evaluate the degree of control we have over that path.

2 Defining A Synthetic Graph Navigation Task

A DAG **G = (N,E)** is made up of set of nodes $N = \{X_i\}_{i=1}^{|N|}$ and set of directed edges across the nodes $E = \{(X_i, X_j)\}_{X_i, X_j \in N}$. The edges of a DAG captured by the **adjacency matrix** A where $A_{ij} = 1$ if $(X_i, X_j) \in E$.

A **directed simple path** is a sequence of distinct nodes of **G** which are joined by a sequence of distinct edges. The first node of a path is referred to as the **start node** and the last node is the **goal node** (Fig. 2). A **cycle** is a sequence of nodes connected by a sequence of edges such that the first and last node are identical. The key characteristic defining DAGs is that they contain no cycles. The adjacency matrix of a DAG can always be rearranged to be upper triangular. **Structure of the DAG:** We find that the structure of the underlying DAG can have a large effect on the usefulness of Step-by-Step Inference. When the DAG is hierarchical, between a start and goal node, nodes in the intermediate layers must be visited SI Fig 7 whereas when the DAG is random, there is a uniform probability that 2 nodes are connected and there is no explicit notion of hierarchy. In both these scenarios, we can define the notion of **path diversity:** between any 2 path-connected nodes, there can be several possible paths. We quantify the path diversity in random and hierarchical graphs in SI Fig 7 and describe their construction in the corresponding SI Sec. **Data generating process for single graph scenarios** We focus on two setups in this work, where one allows for context and one does not. This is intentional so that we can explicitly analyze benefits of stepwise inference in the presence of extraneous context, which may influence a model's internalized knowledge, and hence its execution. **Prompt structure and training data generation** In the single graph setting with underlying DAG G, each prompt is made from a single simple path. Given a start node X_{start} and goal node X_{goal}, the model has to classify whether there exists a path from X_{start} to X_{goal}. We create a pair of tokens path and no-path. We constructed two datasets: one that contains stepwise inference and another that does not. Examples of prompts are provided below. For stepwise inference, the path between the start node X_4 to the goal node X_6 is represented as goal : X_4 X_6 X_5 X_7 X_6 path end. Without

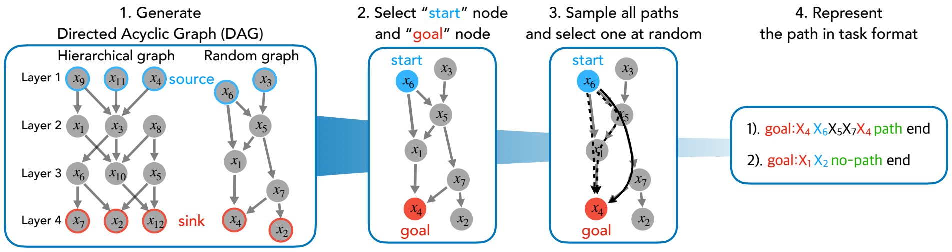


Figure 2: **Data Generating Process for a Single Graph:** This figure illustrates the step-by-step process of generating a training dataset using a single graph. 1) A directed acyclic graph (DAG) is generated, which can be either hierarchically structured or random. 2) A start node and a goal node are selected. 3) All possible paths connecting the start and goal nodes are sampled, and one path is randomly selected. 4) The chosen path is then represented in a task-specific format.

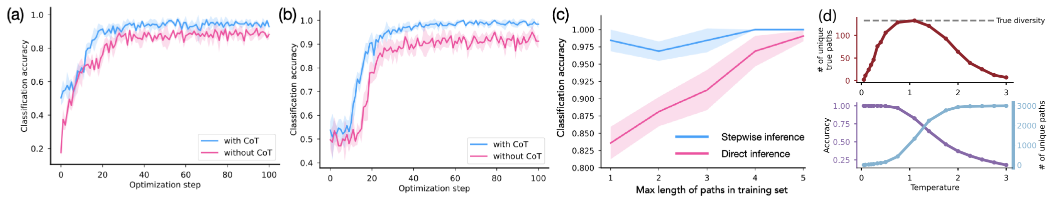


Figure 3: **Advantage of Stepwise Inference in Graph Navigation Tasks:** (a) In random graphs, stepwise inference shows an advantage over direct inference in connectivity prediction tasks. (b) This advantage is further pronounced in hierarchical graphs, where the distances between nodes can be significantly larger. (c) We show that the stepwise inference gap arises when the training set contains paths that are shorter than the paths required to connect nodes in the evaluation set. (d) **A diversity vs. accuracy trade-off in finite temperature stepwise inference for transformers:** As sampling temperature is increased, the diversity of paths generated by the model from a single $(n_{\text{start}}, n_{\text{goal}})$ pair increases, while the accuracy of the path decreases. This tradeoff is captured by measuring the number of unique *true* paths which is non-monotonic (top), showing the existence of an optimal temperature for sampling. The dashed line denotes the ground truth path diversity of $(n_{\text{start}}, n_{\text{goal}})$

stepwise inference, the example path is represented as goal : X_4 X_6 path end. **Results on single-graph scenarios:** Our first result is that a transformer trained on directed edges and a small fraction of node pairs from a fixed underlying DAG can generalize to *all* node pairs, including those held out during training, producing valid simple paths from start to goal nodes. Thus the model can 'stitch' or mix-and-match (sub)paths it has observed over training to produce a valid path across a pair of held-out connected nodes. The training dynamics of a typical network together with a description of failure modes is shown in SI Fig. 8. **A single underlying graph:**

The stepwise inference gap Fig. 3 shows the accuracy of path/no-path classification for (a) a random DAG and (b) a hierarchical DAG. We trained two distinct models using two types of datasets: one with stepwise inference paths and one without. We find that the model trained on the dataset with stepwise inference (represented by the blue line) achieves higher classification accuracy than the model without stepwise inference (the pink line) in both cases. This phenomenon echoes findings from large-scale experiments where the inclusion of intermediate reasoning steps results in increased accuracy (Kojima et al., 2022). We refer to the difference in classification performance with and without stepwise inference as the 'stepwise inference gap'. We also observe that the stepwise inference gap is larger for hierarchical graph than for random graph.

Stitching of paths We hypothesize that stepwise inference is useful when the training data has the following structure: (1) the underlying DAG is hierarchical, which means that there is an explicit feedforward ordering of nodes and to go from nodes in one layer to next one must pass through all intermediate layers and (2) the model must 'stitch' together subsets of paths seen over pretraining in flexible ways to generalize. To test this, we trained the model using paths from hierarchical DAGs while varying the lengths of paths in the training data. Specifically, we created training data that contains start nodes from layer l and goal nodes from layer l' and restricted $l' - l < \Delta$, where Δ denotes the length of the path. During evaluation, we choose node pairs such that $l' - l \geq \Delta$.

The diversity-accuracy tradeoff with higher sampling temperatures Fig. 3d illustrates the effect

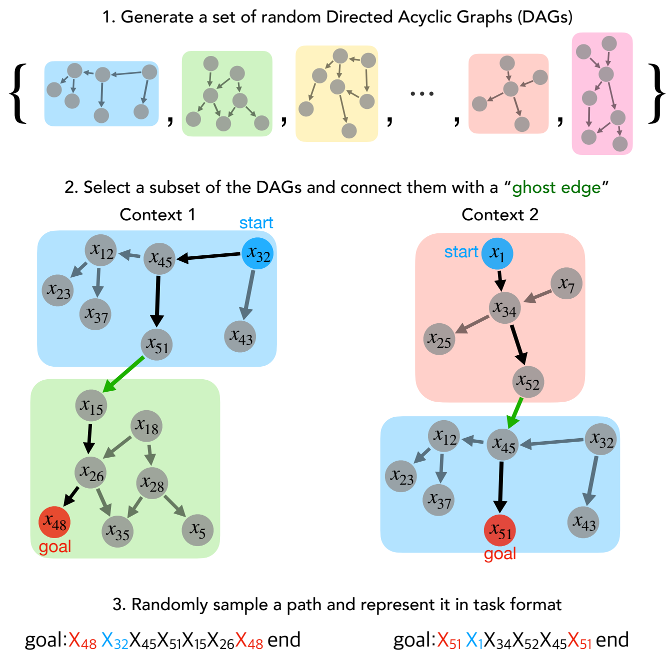


Figure 4: **Data Generating Process for Connected Sub-Graphs (Motifs):** This figure illustrates the step-by-step process of generating a training dataset by combining multiple subgraphs (motifs). 1) We start by making a set of random directed acyclic graphs (DAGs). 2) Next, we pick a subset of these DAGs and connect them together using "ghost edges" to create a bigger graph. 3) From this bigger graph, we randomly sample paths and turn them into a task format.

of sampling temperature on the accuracy and diversity of the generated paths. LLM inference at 0 sampling temperature is equivalent to taking the most likely token at each time step (the maximum likelihood estimate). In this setting, the model deterministically generates the same path for every provided pair of start and goal nodes: n_{start} and n_{goal}. However, in the underlying graph, there exists a diversity of paths from each n_{start} to n_{goal}.

To capture this diversity, we fixed the start node n_{start} and the goal node n_{goal}, and prompted the model 3,000 times, sweeping through different sampling temperatures in Fig. 3d.

Results on multi-graph scenarios

The single graph setting let us explore *zero-shot* planning and stepwise reasoning, where the model relied purely on knowledge internalized over pretraining to do stepwise reasoning. To study how context can influence the path the model traverses, we introduce the concept of motifs and in-context exemplar paths.

Data generating process for the multi-graph scenario To model few-exemplar based chain-of-thought prompting, we modify our single graph setup to include a set of subgraphs that we refer to as **motifs**, denoted by $\{G_i\}_{i=1}^{n}$ A motif is a DAG that is fixed across pretraining and inference. (i) **Ghost edge:** For a pair of connected motifs $G_i \mapsto G_j$, an edge between a sink node of G_i and a source node of G_j, and (ii) A **primitive sequence** (Fig. 4) is sequence of nodes across 2 motifs G_i and G_j with a start node in G_i and goal node in G_j and this sequence contains exactly 1 ghost edge. Fig. 4.

In chain-of-thought prompting (Wei et al., 2022), one or more examples of reasoning are provided before asking the next question, as illustrated in Figure 1(c). The LLM then generate a chain-of-thought which matches that of the exemplar. To model this, we chain a subset of k motifs $G_{c_1} \to G_{c_2} \to ... \to G_{c_k}$ together and provide exemplars Each exemplar e is a primitive sequence across each pair of consecutive motifs: $e \in (G_{c_i} \to G_{c_{i+1}})$ which contains exactly 1 ghost edge. The construction of a primitive sequence is described in Fig. 4 and examples are shown in Fig. 5(b). Given a start node $n_{\text{start}} \in G_{c_1}$ and a goal node $n_{\text{goal}} \in G_{c_K}$, the model can be prompted either directly (Fig. 5(a)) or provided with exemplars and then queried for a path from n_{start} to n_{goal} (Fig. 5(b)). We find that the model can successfully follow the chain defined by the in-context exemplars.

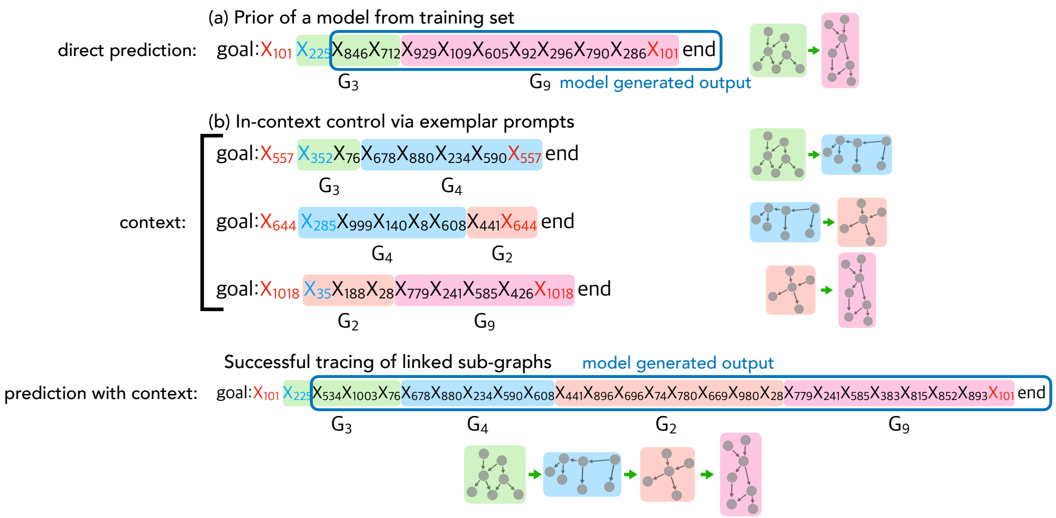


Figure 5: **Example output sequences from the model highlighting the steerability of stepwise inference.** (a) Direction prediction: Given $n_{\text{start}} \in G_3$ and $n_{\text{goal}} \in G_9$, the model produces a path from $G_3 \to G_9$, placing a single ghost edge (X_{712}, X_{929}). (b) With in-context exemplars: primitive sequences from $G_3 \to G_4$, $G_4 \to G_2$ and $G_2 \to G_9$ in-context make the model steer its navigation through the path stringing together these motifs in order: $G_3 \to G_4 \to G_2 \to G_9$, placing a ghost edge between every consecutive motif, for a total 3 ghost edges.

An example output produced by the model is in SI Fig.5, highlighting the path the model takes through the chain of motifs $G_3 \to G_4 \to G_2 \to G_9$. We also find that the model generalizes to arbitrary orders of motifs strung out, including those that did not occur consecutively in the train set – in other words, in-context control is capable of *compositional generalization* (Li et al., 2023).

How do the exemplars affect controllability of graph navigation?

Next, we study how the structural content of the exemplars affects the navigation path chosen by the model. We hope to shed some light on and create hypotheses for the vast and varied findings about stepwise reasoning in LLMs at scale.

Number of intermediate motifs In Fig.6(a), we varied the number of exemplars provided to the model. This is equivalent to stringing together a longer chain of motifs to navigate over. We find that the model can generalize well to unseen orders of motif up to the maximum number chained together in the training data. This creates a hypothesis for chain-of-thought and related methods at scale: the model will fail to generalize to reasoning chains longer than those present in its training data.

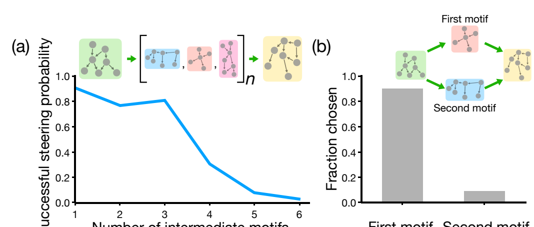


Figure 6: **How do the number of examplers affect the controllability of motifs?** (a) As we vary the number of intermediate motifs in a chain, the path generated by the model follows the path described by the chain until $n = 4$, which is the extent of the training data. (b) In the case of 2 conflicting chains in-context, the model has a bias to pick the first chain.

Bias towards the first exemplar in the case of conflict Multiple examples of a context provided in the prompt can increase the precision of our control over the model, but it can also lead to confusion. Here, we systematically and quantitatively study the behavior of the model when two contexts are provided but are in conflict. In Fig.6(b) we study a scenario where two chains of motifs are provided, starting from the same set of primary motifs and ending at the terminal motif. We find that the model has a strong bias toward choosing the first chain over the second.

References

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

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jian, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. Faith and fate: Limits of transformers on compositionality. *arXiv preprint arXiv:2305.18654*, 2023.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

Yingcong Li, Kartik Sreenivasan, Angeliki Giannou, Dimitris Papailiopoulos, and Samet Oymak. Dissecting chain-of-thought: A study on compositional in-context learning of mlps. *arXiv preprint arXiv:2305.18869*, 2023.

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.

Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=qFVVBzXxR2V`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

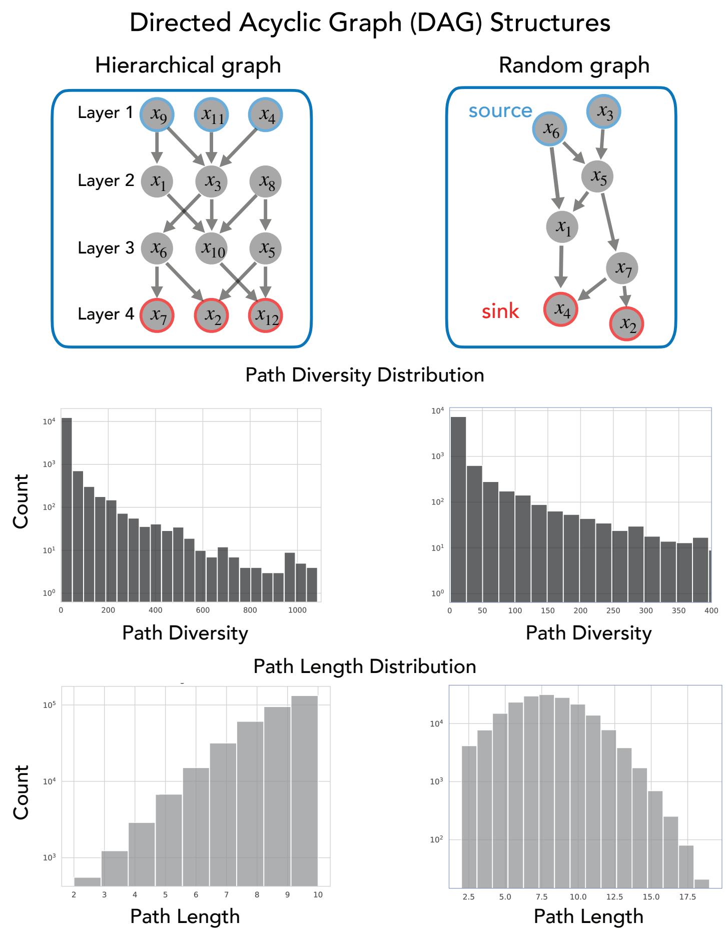


Figure 7: **Construction and properties of Hierarchical and Random DAGs.**

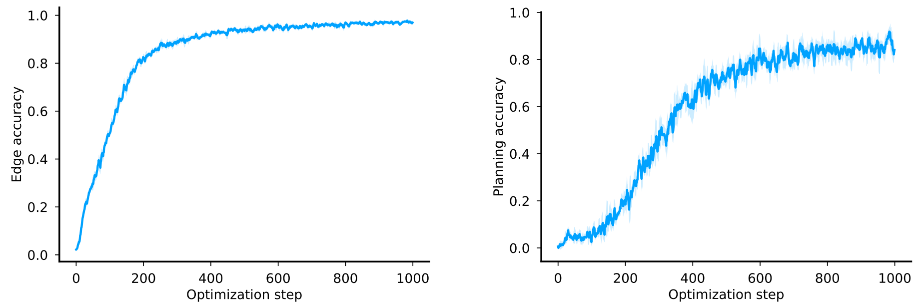
A Appendix

A.1 Setup and construction of graph and model

Here we describe the properties of the DAGs we use, the training setup, model architecture and hyperparameters.

We use 2 DAG structures, hierarchical and random (Fig. 7). Random DAGs are constructed by randomly generating an upper-triangular matrix where each entry has probability p of existing. Hierarchical DAGs are generated by predefining L sets of nodes and drawing an edge between a node n_l in layer l and n_{l+1} in layer $l+1$ with probability p. Lastly, we ensure that the graph is connected. These lead to different path diversity and path length distributions, which affect the efficacy of stepwise inference, as shown in our results.

To create a feedforward hierarchical DAG we construct a set of L layers with N nodes each. For every node n_l in layer l and n_{l+1} in layer $l+1$, we draw a directed edge (n_l, n_{l+1}) with probability p, which we refer to as **edge density**. Thus on average, between any two layers there are pN^2 edges and each node in an intermediate layer has an out-degree and in-degree of pN. The number of paths from a particular node in layer l to layer $l' > l$ is exponential and given by $(pN)^{l'-l}$ - this is quantified in the path length distribution shown in SI Fig 7. Lastly, we make sure that the graph is connected and there no disjoint disconnected components. The nodes from layer 1 are the **source nodes:** nodes $\{X_i\}$ of DAG **G** with parents$(X_i) = \emptyset$ and the nodes from layer L are **sink nodes:** nodes $\{X_i\}$ of DAG **G** with children$(X_i) = \emptyset$.
To create a random DAG of N nodes and create a random upper triangular adjacency matrix $A_{N \times N}$



(a) Missteps (b) Planning failure

Figure 8: **The evolution of failure mode probabilities over training:** It can be seen that the model first learns to produce correct edges (effectively bigram statistics) and then learns the global objective of producing a path that ends at the cued goal node. Accuracy curves averaged over 3 trained models with different random seed.

with bernoulli entries with edge density p, such that $p(A_{ij} = 1) = p$. We also ensure that the graph is connected. This results in a bell-shaped path length distribution, SI Fig.7.

A.2 Training dynamics in the single graph scenario

Here we show the training dynamics of a single graph model.

Failure modes of step-by-step inference Given underlying DAG G, during step-by-step inference, the model produces a sequence of nodes from the start node n_{start} which has to terminate at the goal node n_{goal}, given by the sequence $n_0 = n_{\text{start}} \to n_1 \to n_2 \to ... \to n_k \to ... \to n_T$. Here in our setup, there are two broad categories of failures possible (Saparov & He, 2023):

1. **Misstep:** $(n_k, n_{k+1}) \notin G$. An edge produced by the model does not exist in the DAG.
2. **Planning failure:** $n_T \neq n_{\text{goal}}$. The model produces a path that does not terminate at the goal.

We choose to highlight the two types of failures identified above: (1) the probability of taking a correct step (i.e. $1 - Pr(\text{misstep})$) and (2) the probability of ending at the cued target node (i.e. $1 - Pr(\text{planning failure})$). These are shown in Fig. 8

A.3 Model architecture and loss function

For training, we tokenize every node and we use next-token prediction with a cross entropy loss:

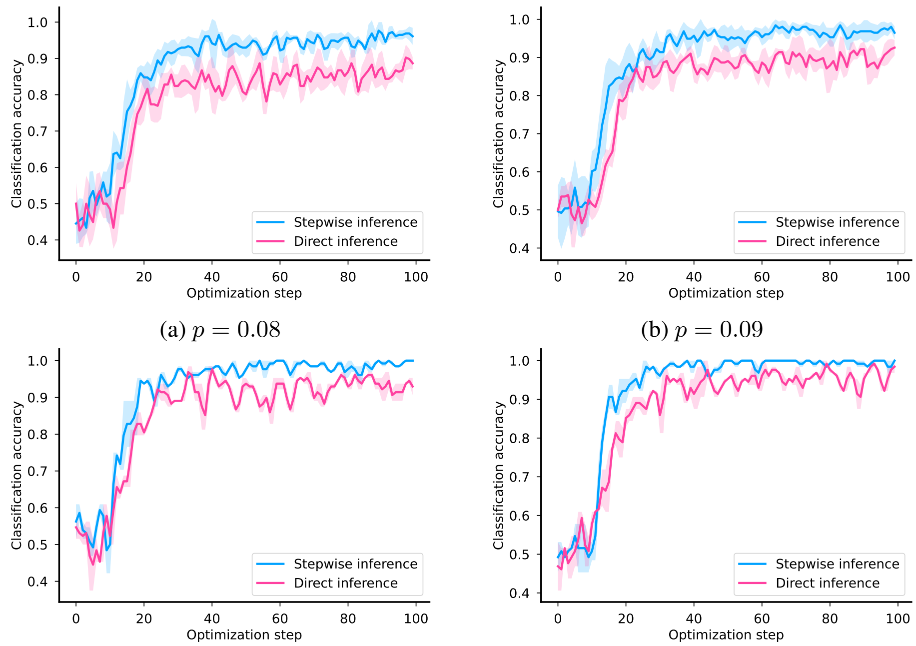
$$\mathcal{L}(\mathbf{x}_n, \text{target } n) = -\log\left(\frac{\exp(\beta x_{n,\,\text{target } n})}{\sum_{t=0}^{\#\text{tokens}} \exp(\beta x_{n,t})}\right) = -\log\Big(\underbrace{\text{softmax}(\beta \mathbf{x}_n)_{\text{target } n}}_{\textbf{prob}(\text{target } n)}\Big) \tag{1}$$

Hyperparameter	Value
learning rate	10^{-4}
Batch size	64
Context length	32
Optimizer	Adam
Momentum	0.9, 0.99
Activation function	GeLU
Number of blocks	2
Embedding dimension	64

Table 1: Hyperparameters of the transformer

For model architecture, we use a GPT based decode-only transformer with a causal self-attention mask. Our implementation is based on the popular nanoGPT repository[1].

[1]available at https://github.com/karpathy/nanoGPT



(a) $p = 0.08$ (b) $p = 0.09$

(c) $p = 0.11$ (d) $p = 0.12$

Figure 10: **Advantage of Stepwise Inference in Graph Navigation Task.**

A.4 A bias towards shorter paths

Fig. 9 examines the average path lengths in a random graph, comparing true paths to those generated by our trained model. Notably, the model consistently produces paths that, on average, are shorter than the actual paths in the random graph. This observation suggests that the model has a bias towards efficiency, which can lead to oversimplification of complex stepwise inference or omission of important intermediate steps.

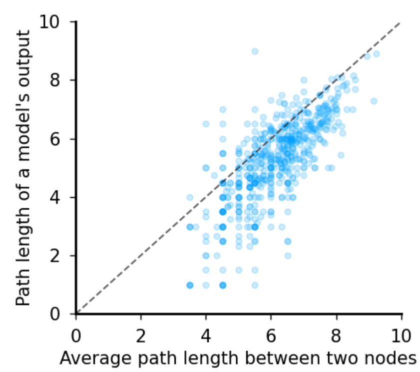


Figure 9: **Model outputs are biased toward shorter paths.** We compare the average lengths of actual and model-generated paths in a random graph, revealing the trained model's bias to generate shorter paths connecting a pair of start and goal nodes in a random graph.

Fig. 4 For the training data construction in the multigraph setting.

A.5 Additional experimental results for the single graph setting

In Fig. 10, we swept the density of the graph from 0.08 to 0.12 on a hierarchical graph. We observe a stepwise inference gap in all cases. The stepwise inference gap becomes smaller for larger densities.

Fig. 11 presents a density plot comparing the average lengths of actual paths with those generated by the model in a random graph. This observation verifies the model tends to produce shorter paths between a given pair of start and goal nodes.

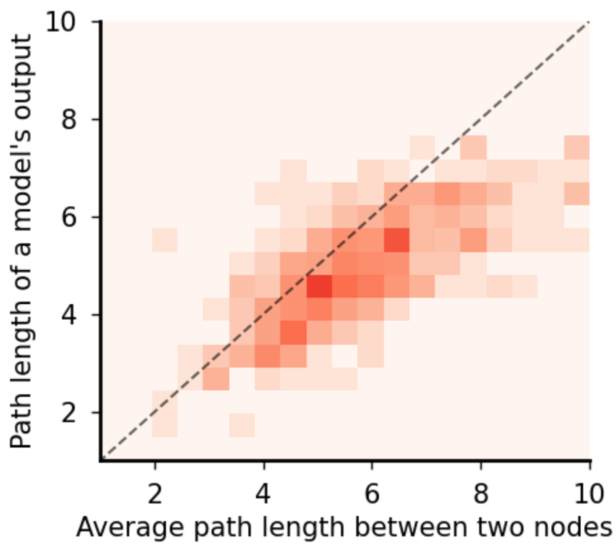


Figure 11: **Model outputs are biased toward shorter paths.**