Towards an Understanding of Stepwise Inference in Transformers: A Synthetic Graph Navigation Model

Anonymous Authors[1]

Abstract

Stepwise inference, such as scratchpads and chain-of-thought (CoT), is an important capability of large language models, where a model decomposes a complex task into a sequence of manageable subproblems. However, despite the significant gain in performance, the underlying mechanisms of stepwise inference have remained elusive. To address this gap, we propose to study auto-regressive Transformer models solving a graph navigation problem, where a model is tasked with traversing a path from a start to a goal node on a synthetically generated graph. Through this simple, controllable, and interpretable framework of graph navigation, we empirically reproduce and analyze several phenomena observed at scale: (i) the stepwise inference reasoning gap, the cause of which we find in the structure of the training data; (ii) a diversity-accuracy tradeoff as sampling temperature varies; (iii) a simplicity bias in the model's output; and (iv) combinatorial generalization, failure on length generalization and a primacy bias with in-context exemplars. Overall, this work introduces a grounded synthetic framework for studying stepwise inference and offers mechanistic hypotheses that lay the foundation for a deeper understanding of this phenomenon.

Stepwise inference, such as scratchpad and chain-of-thought (CoT), is an important capability of large language models, where a model decomposes a complex task into a steps of manageable subproblems. However, despite significant improvements in performance, the underlying mechanisms of these processes have remained elusive. Here, we propose a novel perspective by framing the tasks that benefit most from stepwise inference as graph navigation problems. Specifically, we introduce a graph navigation task, where a transformer model trained from predicts the connectivity between two nodes within a well-defined graph. Despite its simplicity, we demonstrate that this simple setup allows us to not only replicate, but also elucidate behaviors associated with stepwise inference, including the influence of the underlying graph's structure on model behavior, and the trade-off between diversity and accuracy. In addition, we investigate the 'simplicity bias' observed in stepwise inference, where models exhibit a preference for the shortest path between two points. Our investigation extends to the controllability of these models through in-context exemplars. By developing a grounded synthetic framework, this work contributes to the understanding of stepwise inference and provides mechanistic hypotheses that pave the way for a more comprehensive understanding of these phenomena in large-scale language models.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

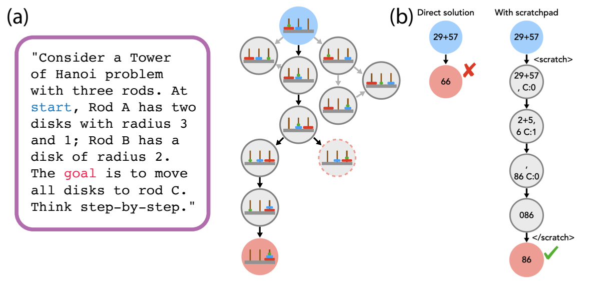

"Consider a Tower of Hanoi problem with three rods. At start, Rod A has two disks with radius 3 and 1; Rod B has a disk of radius 2. The goal is to move all disks to rod C. Think step-by-step."

Figure 1. **Examples of stepwise inference protocols and how they can be cast as a graph navigation problem.** (a) Zero-shot chain-of-thought (Kojima et al., 2022) involves asking a model to produce intermediate outputs to perform complex multi-step computations, such as solving the Tower of Hanoi problem. Casting the configurations of the rods in Tower of Hanoi as nodes of a graph, we can see that the problem is essentially traversal over states describing different configurations of the setup to reach the desired configuration (the goal state). (b) Scratchpad (Nye et al., 2021) improves LLMs' ability to perform complex multi-step computations, such as arithmetic, when they write intermediate computation steps to a buffer called a scratchpad.

1. Introduction

Transformers, the backbone of large language models (LLMs), have revolutionized several domains of machine learning (OpenAI, 2023; Anil et al., 2023; Gemini et al., 2023; Touvron et al., 2023). An intriguing capability that emerges with training of Transformers on large-scale language modeling datasets is the ability to perform **stepwise inference**, such as *zero-shot chain-of-thought (CoT)* (Kojima et al., 2022), use of *scratchpads* (Nye et al., 2021), *few-shot CoT* (Wei et al., 2022), and variants of these protocols (Creswell et al., 2022; Yao et al., 2023; Besta et al., 2023; Creswell & Shanahan, 2022; Press et al., 2022). Specifically, in stepwise inference, the model is asked to or shown exemplars describing how to *decompose a broader problem into multiple sub-problems*. Solving these sub-problems in a *step-by-step* manner simplifies the overall task and significantly improves performance (see Fig. 1). Arguably, stepwise inference protocols are the workhorse behind the "sparks" of intelligence demonstrated by LLMs (Bubeck et al., 2023)—yet, their inner workings are poorly understood.

Motivated by the above, we aim to design and study an abstraction which enables a precise understanding of stepwise inference in Transformers. Specifically, we argue that tasks which see maximum benefit from stepwise inference can be cast as a **graph navigation** problem: given an input describing the data to operate on and a goal to be achieved, a sequence of primitive skills (e.g., ability to perform arithmetic operations) is chained such that each skill acts on the previous skill's output, ultimately to achieve the given goal. If the input data, the final goal, and the sequence of intermediate outputs are represented as a sequence of nodes of a graph, along with primitive skills as edges connecting these nodes, the overall task can be re-imagined as navigating nodes of the graph via the execution of primitive skills. Several logical reasoning problems come under the purview of this abstraction (LaValle, 2006; Cormen et al., 2022; Momennejad et al., 2023; Dziri et al., 2023; Saparov & He, 2023): e.g., in Fig. 1a, we show how the problem of Tower of Hanoi can be decomposed into simpler sub-problems. See also Appendix A for several more examples.

This work. We design a graph navigation task wherein a Transformer is trained from scratch to predict whether two nodes from a well-defined graph can be connected via a path. A special prefix indicates to the model whether it can generate intermediate outputs to solve the task, i.e., if it can generate a sequence of nodes to infer a path connecting the two nodes; alternatively, exemplars demonstrating navigation to "regions" of the graph are provided. Our framework assumes the model has perfect skills, i.e, any failures in the task are a consequence of incorrect plans for navigating the graph. This is justified because a skill-based failure is the most trivial mechanism via which stepwise inference protocols can fail; in contrast, inability to plan is an independent and underexplored axis for understanding stepwise inference. Overall, we make the following contributions.

- **A Framework for Investigating Stepwise Inference.** We propose a synthetic graph navigation task as an abstraction of scenarios where stepwise inference protocols help Transformers improve performance, showing that we can replicate and explain behaviors seen with use of stepwise inference in prior work. For instance, the structure of the data generating process (the graph) impacts whether stepwise inference will yield any benefits (Prystawski & Goodman, 2023). We identify further novel behaviors of stepwise inference as well, such as the existence of a tradeoff between diversity of outputs generated by the model and its accuracy with respect to inference hyperparameters (e.g., sampling temperature).
- **Demonstrating a Simplicity Bias in Stepwise Inference.** When multiple solutions are possible for an input, we demonstrate the existence of a *simplicity bias*: the model prefers to follow the shortest path connecting two nodes. We assess this result mechanistically by identifying the underlying algorithm learned by the model to solve the task, showing the bias is likely a consequence of a "pattern matching" behavior that has been hypothesized to cause LLMs to fail in complex reasoning problems (Dziri et al., 2023).
- **Controllability via In-Context Exemplars.** We show the model's preferred path to navigate between two nodes can be controlled via use of in-context exemplars. We use this setup to evaluate the model's ability to generalize to paths of longer length and the influence of exemplars which conflict with each other, i.e., that steer the model along different paths.

2. Stepwise Inference as Graph Navigation

In this section, we define our setup for studying how stepwise inference aids Transformers in solving complex reasoning problems. Specifically, we define a graph navigation task wherein, given a start and a goal node, a Transformer is autoregressively trained to produce a sequence of nodes that concludes at the goal node. In our experiments, we consider two scenarios: one where in-context exemplars are *absent* (see Fig. 2a) and another where they are *present* (see Fig. 2b). The former scenario emulates protocols such as the scratchpad and zero-shot Chain of Thought (CoT) (Kojima et al., 2022; Nye et al., 2021), while the latter models few-shot CoT (Wei et al., 2022). In Section 2.1, we set up our experiment to explore these two scenarios. In the subsequent sections, we explicitly analyze the benefits of stepwise inference in both scenarios: without in-context exemplars (Section 2.2) and with in-context exemplars (Section 2.3). We refer the reader to a detailed related work on stepwise inference protocols in Appendix B and further discussion on graph navigation as a model of stepwise inference which is in Appendix A.

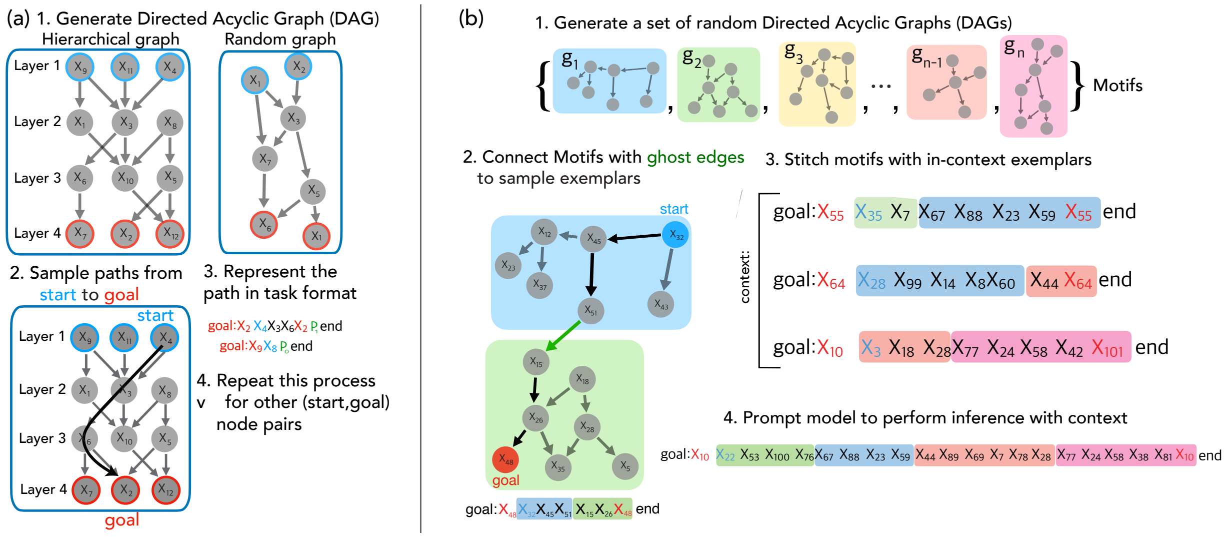

(a) 1. Generate Directed Acyclic Graph (DAG)
Hierarchical graph
Random graph
Layer 1
Layer 2
Layer 3
Layer 4
2. Sample paths from start to goal
3. Represent the path in task format
goal:X2 X4X3X6X2 P end
goal:X9X8 P end
4. Repeat this process for other (start,goal) node pairs
start
goal
(b)
1. Generate a set of random Directed Acyclic Graphs (DAGs)
g1, g2, g3, ..., gn-1, gn
Motifs
2. Connect Motifs with ghost edges to sample exemplars
3. Stitch motifs with in-context exemplars
context:
goal:X55 X35 X7 X67 X88 X23 X59 X55 end
goal:X64 X28 X99 X14 X8X60 X44 X64 end
goal:X10 X3 X18 X28X77 X24 X58 X42 X101 end
4. Prompt model to perform inference with context
goal:X10 X22 X53 X100 X76X67 X88 X23 X59 X44 X89 X69 X7 X78 X28 X77 X24 X58 X38 X81 X10 end
goal:X48 X32X45X51 X15X26X48 end

Figure 2. **Data Generating Process.** (a) **In absence of exemplars.** This figure illustrates the step-by-step process of generating a training dataset using a single underlying graph. 1) A directed acyclic graph (DAG) is generated, which can be either hierarchically structured or Bernoulli. 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. (b) **In presence of exemplars.** The step-by-step process of generating a training dataset by combining multiple subgraphs (motifs): 1. We start by building a set of Bernoulli directed acyclic graphs (DAGs), 2. Next, we pick a subset of K of these DAGs $\{g_{i_1}, g_{i_2}, ..g_{i_K}\}$ and connect them together using "ghost edges" to create a chain of motifs $g_{i_1} \mapsto g_{i_2} \mapsto .. \mapsto g_{i_K}$, 3. We then sample exemplars from every pair of motifs that have been connected by a ghost edge to construct the context and 4. We now choose a start node $X_s \in g_{i_1}$ and a goal node $X_g \in g_{i_K}$ and construct a sequence passing through the whole motif chain.

2.1. Preliminaries: Bernoulli and Hierarchical DAGs

We use **directed acyclic graphs (DAGs)** to define our graph navigation tasks. DAGs are a natural mathematical abstraction to study multi-step, logical reasoning problems: e.g., as discussed in Dziri et al. (2023), the output of any deterministic algorithm can be represented as a DAG. Specifically, a DAG is defined as $G := (N, E)$, where $N := \{X_i\}_{i=1}^{|N|}$ denotes the set of nodes in the graph and $E := \{(X_i, X_j)\}_{X_i, X_j \in N}$ denotes the set of directed edges across the nodes. The edges of a DAG are captured by its *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 edges. If two nodes are connected via a directed simple path, we call them **path-connected**. The first node of a path is referred to as the start node, which we denote as X_s, and the last node as the goal node, which we denote as X_g.

We briefly discuss the process of construction of DAGs used in our work and how paths are sampled from them; a more thorough description is provided in Appendix C.1. We define a **Bernoulli DAG** of N nodes, whose adjacency matrix has an upper triangular structure with Bernoulli entries with edge density p, such that $p(A_{ij} = 1) = p$. We ensure that all nodes have at least one edge (see Fig. 2a). The resulting DAG exhibits a bell-shaped path length distribution (see Fig. 11 in Appendix C.1). We also define a **hierarchical DAG**, wherein the nodes follow a feedforward, layered structure such that all nodes at a given layer are only connected to nodes in the following layer (see Fig. 2a). In particular, 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**. On average, between any two layers of a hierarchical DAG, 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 Appendix Fig. 11. For both graph structures, **source nodes** are nodes that do not have any parent nodes, and the nodes that do not have any children nodes are **sink nodes**.

2.2. Modeling stepwise inference without exemplars

Zero-shot CoT (Kojima et al., 2022) and scratchpads (Nye et al., 2021) represent two examples of stepwise inference protocols that do not rely on exemplars. For instance, in the zero-shot CoT approach, the input of the model is augmented with the phrase `let's think step by step`. This encourages the model to generate outputs that elaborate on the intermediate steps required to solve the target problem, thereby enhancing accuracy by breaking down the target problem into several simpler problems.

To compare the model's performance with stepwise inference and without stepwise inference (i.e., direct inference) in such scenarios, we create two datasets: one including intermediate steps and the other without them. Each dataset was subsequently used to train distinct transformers. During the testing phase, we presented the model with pairs of nodes and task it to determine the existence of a path between them. The model's performance is assessed based on their accuracy in classifying whether a path exists.

Fig. 2a shows how we generate the datasets above. First, we define a DAG denoted as G. Within this graph, for each dataset instance, we sample a start node X_s and a goal node X_g and then identify all feasible paths between these two nodes. From the identified paths, we select one to form a sequence of tokens, S. This procedure is iterated for other node pairs within the graph G to compile the complete dataset. For the dataset with stepwise inference, we use all the intermediate steps, including the start node X_s and the goal node X_g, to form S. For the dataset without stepwise inference (i.e., direct inference), we only use the start node X_s and the goal node X_g. We introduce a binary variable $\texttt{path} \in \{\texttt{p}_1, \texttt{p}_0\}$ to denote whether there is a path between the start and goal nodes. We append the 'path' token $\texttt{p}_1$ to the end of the sequence S if there is at least one path between the start and goal nodes; otherwise, we append the 'no path' token $\texttt{p}_0$.

Example: For the example path in Fig. 2a, in the dataset with stepwise inference, the sequence of tokens S includes the intermediate steps and takes the form $\texttt{goal}:X_2, X_4, X_3, X_6, X_2, \texttt{p}_1$. For the dataset without stepwise inference (i.e., direct inference), the sequence S does not contain intermediate steps and has the form $\texttt{goal}:X_2\ X_4\ \texttt{p}_1$.

2.3. Modeling stepwise inference with exemplars

Here we examine the influence of stepwise inference on model performance when in-context exemplars are present. This scenario is prominently exemplified by protocols based on few-shot CoT (Wei et al., 2022; Creswell et al., 2022).

Specifically, we extend the setup with a single DAG described in Section 2.2 by incorporating a *set of DAGs*, which we call **motifs**. The data generation process is shown in Fig. 2b. First, we generate a set of n Bernoulli DAGs denoted by $\widehat{g} = \{g_i\}_{i=1}^{n}$ and randomly select a subset of K motifs from this set: $\{g_{j_1}, g_{j_2}, \ldots, g_{j_K}\}$. Second, we add edges between the sink node of each motif g_{j_k} and the source node of the subsequent motif $g_{j_{k+1}}$, forming a chain of motifs $g_{i_1} \mapsto g_{i_2} \mapsto \cdots \mapsto g_{i_K}$. These interconnecting edges are termed **ghost edges**. Third, we sample paths from each pair of motifs linked by a ghost edge to establish the context. We then select a start node from the sink nodes of one motif, $X_s \in g$, and a goal node from the source nodes of a different motif, $X_g \in g'$, then sample a path between them, denoted as $e_{gg'}$. This procedure generates a sequence of nodes spanning across motifs, $g \to g'$, including exactly one ghost edge. We refer to this as an **exemplar sequence** and use them as in-context samples. Exemplars to model few-shot CoT are represented as $e_{gg'}$ and denote a exemplar sequence from the motif $g \to g'$. Finally, we choose a start node $X_s \in g_{i_1}$ and a goal node $X_g \in g_{i_K}$. We then prompt the model either directly to output a path that connects the pair of nodes or exemplars demonstrating a traversal between motifs of the graph G can be provided in context (see Fig. 2b). Since a graph is defined via combinations of motifs, we intentionally leave out 20% combinations from the training data. Every time a new input is to be designed, we will randomly select motifs from the remaining combinations from $\widehat{g}$ to design a graph G, sampling sequences showing how to travel between two nodes from this graph. Though a model can learn the structure of the motifs themselves and how some of these motifs are connected, the graphs at test time will involve combinations of motifs that were never seen in training. Correspondingly, *the model must use the context to infer the structure of the overall graphs*. In essence, an exemplar tells the model which motifs are connected via ghost edges and hence can be navigated between.

Example: We directly study the path of navigation outputted by the model in this setup, i.e., no special tokens are used. A sample is constructed by selecting motifs to define in-context exemplars, say $g_{i_1}, g_{i_2}, g_{i_3}$. For every successive pair of motifs, we construct an exemplar and put them together to create the context. To do this, we select two (start, goal) pairs: $X_{s_1} \in g_{i_1}$, $X_{g_1} \in g_{i_2}$ and $X_{s_2} \in g_{i_2}$, $X_{g_2} \in g_{i_3}$. We sample exemplar sequences starting and ending at these node pairs: one sequence from g_{i_1} to g_{i_2}, $\texttt{goal}:X_{g_1} X_{s_1} X_1 \ldots X_{k_1} X_{g_1}$, and another from g_{i_2} to g_{i_3}, $\texttt{goal}:X_{g_2} X_{s_2} X'_1 \ldots X'_{k_2} X_{g_2}$. These sequences act as exemplars to be provided in context to the model when it is shown an input. The number of exemplars can vary from 2 to 4, which correspond to chains of motifs of length 3 to 5. The input itself is defined by choosing a goal node $X_g \in g_{i_3}$, a start node $X_s \in g_{i_1}$, and a path through an intermediate node $X_{\text{inter}} \in g_{i_2}$: $\texttt{goal}:X_g X_s X''_1 \ldots X_{\text{inter}} \ldots X''_{k_1} X_{g_3}$. Here, $X_s X''_1 \ldots X_{\text{inter}}$ is a path between motifs g_{i_1} and g_{i_2}, while $X_{\text{inter}} \ldots X''_k X_{g_3}$ is a path between motifs g_{i_2} and g_{i_3}. When exemplars are not provided, the model must rely on its internalized knowledge to infer whether there exist two connected motifs that can be used to move from the start to goal node. The context exemplars simplify the problem by telling the model the motifs above are connected.

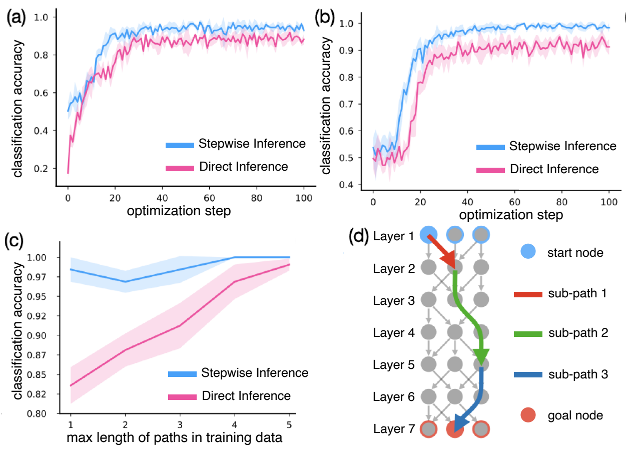


Figure 3. **Advantage of stepwise inference in graph navigation tasks and stitching:** (a) In the Bernoulli DAG, stepwise inference demonstrates an advantage over direct inference in predicting whether given node pairs are connected. (b) This advantage is further pronounced in hierarchical DAGs, where the distances between nodes are greater than in Bernoulli DAGs. (c) The stepwise inference gap arises when the training set contains paths that are shorter than the paths required to connect nodes at test time. (d) The stepwise inference is beneficial when the model must connect paths seen during training: The red, green, and blue paths represent subsets of paths seen during training; we ask the model to produce paths that combine these paths during the test phase.

3. Results: Stepwise Navigation

In this section, we discuss findings on how stepwise inference affects the model's ability to solve problems. We investigate two scenarios: in the absence of in-context exemplars (Section 3.1) and in the presence of them (Section 3.2). For all experiments, unless stated otherwise, we use a 2-layer Transformer defined by Karpathy (2021) to mimic the GPT architecture (Brown et al., 2020). For more details on the experimental setup, please refer to Appendix C.3 for model architecture details and Appendix D for training data generation and test/train split.

3.1. Navigation without exemplars

3.1.1. Stepwise inference gap

We assess the performance of the model by evaluating its ability to classify whether there is a path given a pair of nodes during the test phase. Specifically, we randomly sample pairs of start and goal nodes that were not seen in the training data and observe whether the model outputs either the 'path' token $\mathtt{p}_1$ or the 'no path' token $\mathtt{p}_0$. Fig. 3 shows the accuracy of classifying 'path' or 'no path' for two different types of graphs: a random graph and a hierarchical graph. We observe that for both types of graphs, the use of stepwise inference significantly improves the model's performance compared to direct inference, with more pronounced improvements noted for the hierarchical graph.

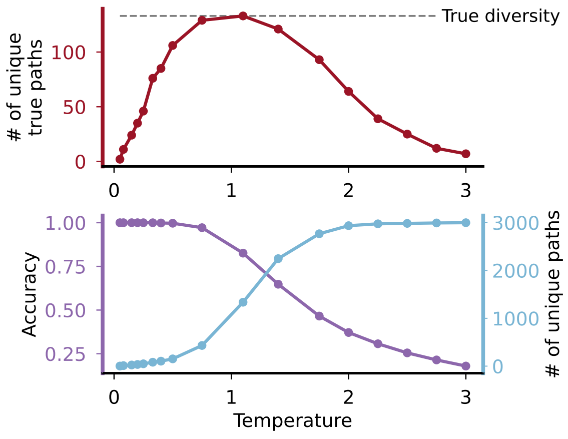


Figure 4. **Diversity vs. accuracy trade-off for different sampling temperatures of the transformer model:** As the sampling temperature increases, the diversity of paths generated by the model also increases, while the accuracy decreases. This tradeoff is captured by measuring the number of unique *valid* paths (top panel), indicating that there is an optimal temperature for sampling. The dashed line represents the ground truth path diversity.

Following Prystawski & Goodman (2023), we refer to the improvement in performance observed between stepwise inference and direct inference as the "stepwise inference gap". To simulate the effect of noisy real-world labels, we introduced random corruption into the tokens and found that these results hold, as detailed in Appendix Fig. 13.

To further probe these results, we control for path lengths in the hierarchical graph. Specifically, to set the maximum path length in the training data to Δ, we choose a starting layer l and a goal layer l' such that $l' - l < \Delta$. Then, we sample starting nodes from layer l and goal nodes from layer l'. For the test data, we select node pairs with $l' - l \geq \Delta$. Results are shown in Fig. 3(c). We plot the classification accuracy across various values of Δ and observe that the smaller the value of Δ, the greater the stepwise inference gap becomes. We hypothesize this happens because when the training data only includes short paths, the model needs to more effectively 'stitch' the paths observed during training, which, as a recursive task, is more feasible via stepwise inference.

3.1.2. Diversity-accuracy tradeoff with higher sampling temperatures

Here, we investigate how the sampling temperature of the autoregressive transformer affects the diversity of the generations produced by the model and its accuracy. To this end, we fixed the start and goal nodes and prompted the model 3,000 times, varying the sampling temperatures from 0.0 to 3.0. We define accuracy as the probability that a generated path consists of valid edges and correctly terminates at the designated goal node. Diversity is defined as the number of unique paths generated. As shown in Fig. 4, there is a

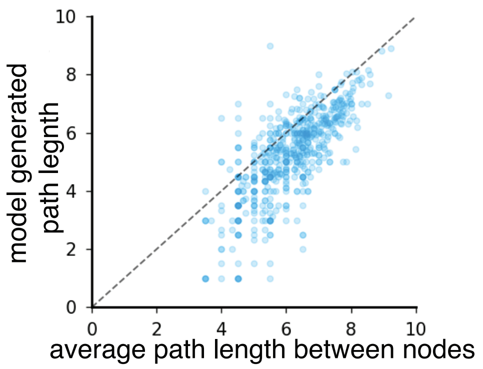


Figure 5. **Model outputs are biased toward shorter paths.** We compared the average lengths of ground-truth paths for a specific set of node pairs and the paths produced by the model for these same pairs in the Bernoulli DAG. We observe that the model tends to generate shorter paths than the actual ones. This observation points to a bias in the trained model towards favoring shorter over potentially more accurate or realistic paths.

clear trade-off between the diversity of the paths generated by the model and their accuracy. We term this phenomenon the ***diversity-accuracy tradeoff***: at lower sampling temperatures, the model generates fewer but more accurate and valid paths; in constrast, higher sampling temperatures result in greater path diversity but reduced accuracy. Our result provides the first explicit demonstration of a trade-off between the accuracy and diversity of transformer outputs. To the best of our knowledge, this phenomena has not been quantitatively studied before.

3.1.3. Preference for shorter paths

As mentioned before, there are multiple possible paths the model can choose from in the pursuit of inferring a path that connects a start and goal node. We showed that by increasing the sampling temperature, a diverse set of paths can be generated; however, by default, which paths does the model prefer? To evaluate this, we compare the actual path lengths between nodes in the test data with those generated by our trained model in the Bernoulli graph setup. In Fig. 5a, we observe that the model consistently produces paths that are shorter, on average, than the paths in the ground truth DAG. This observation suggests that the model exhibits a *simplicity bias*, tending to find the quickest path to solve the target problem. Such simplicity biases can lead to the oversimplification of problems (Shah et al., 2020; Lubana et al., 2023).

3.1.4. Evolution of failures in stepwise inference over training

In the above discussion, we evaluated how stepwise inference assists a model in successfully completing a complex, multi-step task. We now assess how it fails. Specifically,

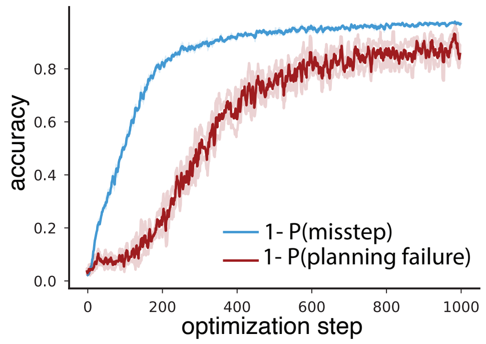


Figure 6. **Learning dynamics for two failure modes: misstep and planning failure.** We measure the probability of missteps and planning failures in the model's outputs. A misstep refers to an instance where the model generates an edge that is not present in the DAG G, while a planning failure means that the model outputs a path that fails to reach the intended goal node. Initially, the model learns to avoid missteps. Subsequently, around the optimization step of 200, it begins to effectively learn planning. The accuracy curves are averaged over three models, each trained with a distinct random seed.

assume that for a given graph G, the model produces a sequence of nodes $X_s X_1 \dots X_k \dots X_t$ starting at the start node X_s. Following (Saparov & He, 2023; Momennejad et al., 2023), we define two categories of potential failures.

- **Misstep** $(X_k, X_{k+1}) \notin G$: An edge produced by the model does not exist in the DAG, commonly referred to as “hallucinations”.
- **Planning failure** $X_t \neq X_g$: The model produces a path that does not terminate at the goal node.

In Fig. 6, we examine the learning dynamics for each failure mode. The figure indicates that the model initially acquires the skill to circumvent missteps (the blue line). Subsequently, it develops the ability to plan effectively, which is shown by a decrease in planning failures (the red line). By integrating these abilities—avoiding missteps and minimizing planning failures—the model is finally able to generate accurate paths for node pairs not seen during training.

3.1.5. Mechanistic basis of the learned graph navigation algorithm

Our results above elicit several intriguing behaviors attributable to stepwise inference. We next take a more mechanistic lens to explain why these behaviors possibly occur. We hypothesize that the model learns embeddings for the nodes of the graph that enable easy computation of an approximate distance measure. This suggests that to move closer to the goal node, one can simply transition to the node that exhibits the least distance from the goal node. For the detailed intuition guiding our analysis, see Appendix F.

To verify this, we first strip the model down to a single-head, self-attention layer. We visualize the attention scores for this

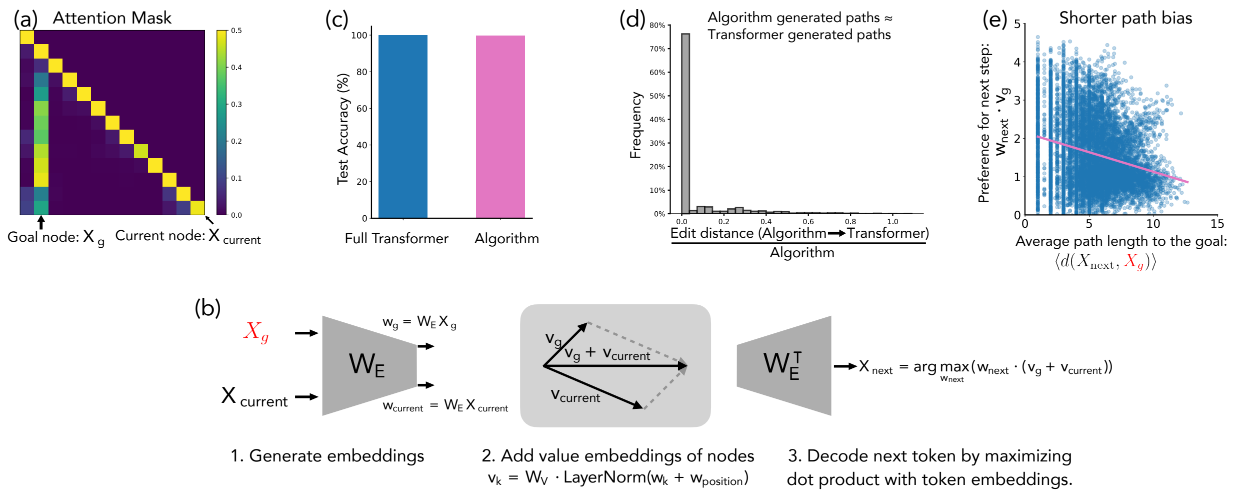


Figure 7. **Mechanistic analysis of the graph navigation algorithm: Emergent linear representation.** (a) Attention maps from the 1-layer, attention-only transformer, highlighting the model's attention on the goal token X_g and the current token X_{current}. (b) Steps of our simplified algorithm that emulates the 1-layer, attention-only transformer: 1. We extract value embeddings for X_g and X_{current}, ignoring other tokens for simplicity. 2. Next, we compute the value embeddings of the goal v_g and current v_{current} nodes and add them together $v = v_g + v_{\text{current}}$. 3. We then compute the token embedding with the highest inner product with v, approximating the token that receives the highest logit score after the single forward pass. (c) Comparison of the model's accuracy on a set of 500 held-out node pairs (X_s, X_g) using our simplified algorithm (99.8%) versus the full trained model (99.6%). (d) The paths generated by the simplified algorithm almost exactly match the paths generated by the full trained model. Path similarity on 2000 held-out node pairs was compared by measuring the Levenshtein edit distance (Navarro, 2001) between paths generated by the full trained model and the simplified algorithm for the same (X_s, X_g) pairs. (e) The short path bias can be attributed to the inner products between the token embedding of the next chosen token X_{next} and the value embedding of X_g and v_g. We observe that nodes X_{next} further away from X_g have a lower inner product, indicating that the model's embedding of nodes reflects the underlying graph topology. The red line denotes the best least squares fit and has a slope of -0.106.

minimal model in Fig. 7a, observing that the attention scores are concentrated on the goal node and the current node. This suggests that the model utilizes only the embedding values of the goal X_g and the current nodes X_{current} to select the next token. Inspired by this observation, we develop a simplified algorithm that mimics the behavior of the model, as outlined in detail in Fig. 7b.

In Fig. 7c, we demonstrate this simplified algorithm matches the accuracy of the full trained model (i.e., the single-head, self-attention layer transformer). Further, in Fig. 7d, we find that the paths generated by our simplified algorithm and those produced by the full trained model are nearly identical. We use a string edit distance metric (Navarro, 2001) to quantify the similarity between the two sets of paths and find that over 75% of paths are identical.

Given that accuracy is computed over test nodes not seen in the training data, it is likely that the model encodes a notion of distance between two nodes on the graph in its embedding, as we hypothesized. Indeed, in Fig. 7e, we find that the inner product of the embedding of v_g with the token embeddings of X_{next} is negatively correlated with the distance between these two nodes in the ground truth DAG. Here, we used the average path length as a distance measure over the graph. Since potential nodes with shorter paths to the goal node have a higher logit value, this implies they will be more likely to be predicted, thus showing the origin of the short path bias we observed in Sec. 3.1.3. This is a mechanistic explanation of the pattern-matching behavior of Dziri et al. (2023) in the context of our task.

3.2. Navigation with exemplars

The single graph setting let us explore *zero-shot* navigation and stepwise reasoning, where the model relied on knowledge internalized over pretraining for stepwise navigation towards a goal. Next, we study how context can influence the model generated paths, how subgoals that are provided in-context can guide the model's navigation, and how the content of the exemplars affects the navigation path chosen by the model. Our results shed some light on and create hypotheses for (1) compositional generalization, (2) length generalization, and (3) impact of conflicting, long context.

3.2.1. Compositional generalization

We find that the model can successfully follow the chain defined by the in-context exemplars. An example output produced by the model is in Fig.2(b), highlighting the path

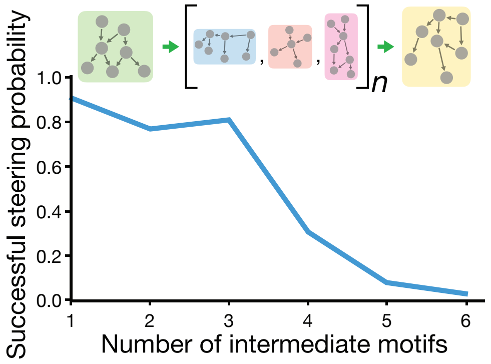


Figure 8. **In-context steerability and length generalization.** We vary the number of intermediate motifs g_{inter} in a chain of motifs constructed for the particular context $g_{i_1} \to \{g_{\text{inter}} \to\}_n \to g_{i_K}$. The path generated by the model follows the path described by the chain in context until $n = 4$, which is the maximum chain length in the training data.

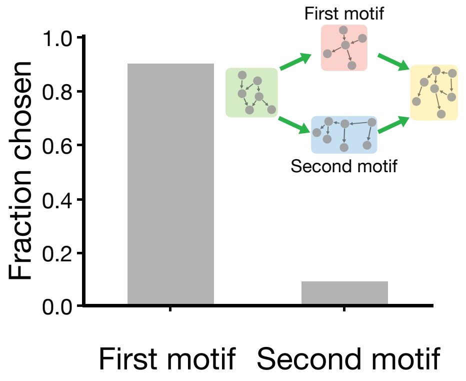


Figure 9. **How does the model handle conflicting exemplars?** To construct the context, we selected an initial motif g_{i_1}, a terminal motif g_{i_T} and two intermediate motifs g_{inter} and g'_{inter}. We string them together so that the motif has two possible paths: $g_{i_1} \to g_{\text{inter}} \to g_{i_T}$ and $g_{i_1} \to g'_{\text{inter}} \to g_{i_T}$. In this case of 2 conflicting chains in-context, the model has a bias to pick the chain that appears first in context.

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 training data, up to the length in the training data (see Fig. 8). In other words, in-context control is capable of eliciting ***compositional generalization*** (Li et al., 2023), if appropriately trained. Further, we see that the attentional patterns used by the model suggest that while navigating across motifs, the model treats nodes across ghost edges as subgoals (see Appendix Fig. 17).

3.2.2. Number of intermediate motifs

In Fig. 8, we vary the number of exemplars provided to the model. This is equivalent to stringing together a longer chain of exemplar sequences across motifs to navigate over. We define successful steering as the product of indicators that the path ended at the specified goal: $\mathcal{I}_{X_t == X_g}$ and that each ghost edge, and thus the intermediate motif, was present in the path: $\mathcal{I}_{g_{\text{inter}}}$. We computed the probability by averaging over distinct source nodes $\in g_{i_1}$ and sink nodes $\in g_{i_K}$. We find that the model can generalize well to unseen orders of motifs up to the maximum number chained together in the training data, after which the model fails to navigate. We hypothesize that even when using stepwise inference methods at scale, *the model will fail to generalize to reasoning chains longer than those present in its training data.*

3.2.3. Bias towards the first exemplar in the case of conflict

In many scenarios, language models are prompted with several examples and has to choose between them based on information in-context, for example in a multiple choice Q&A tasks (Hendrycks et al., 2020; Pal et al., 2022; Srivastava et al., 2022). Here, we systematically and quantitatively study the behavior of the model when two contexts are provided but are in conflict. In Fig. 9, to model scenarios with conflicting exemplars, we study a case where two chains of motifs are provided, starting from the same set of initial and terminal motifs g_{i_1} and g_{i_T}, but with distinct intermediate motifs g_{inter} and g'_{inter}. The model is then prompted with $X_s \in g_{i_1}$ and $X_g \in g_{i_T}$, after in-context exemplars in order: $e_{g_{i_1}, g_{\text{inter}}}, e_{g_{\text{inter}}, g_{i_T}}, e_{g_{i_2}, g'_{\text{inter}}}, e_{g'_{\text{inter}}, g_{i_T}}$. We find that the model does navigate to the goal, thus following the prompt, but has a strong bias toward choosing a path defined by the first chain over the second, i.e., $g_{i_1} \to g_{\text{inter}} \to g_{i_T}$ (see Fig. 9). This result is similar to what happens at scale with large context windows, where content in the middle of a long context window is ignored (Liu et al., 2023).

4. Conclusion

In this work, we introduced a synthetic graph navigation task to investigate the behavior, training dynamics and mechanisms of transformers under stepwise inference protocols. Despite its simplicity, our synthetic setup has provided key insights into the role of the structural properties of the data, the diversity-accuracy tradeoff in sampling, and the simplicity bias of transformer optimization. In addition, we explored the model's navigation preferences and their controllability through in-context exemplars, modeled length generalization, and responses to longer contexts with conflicting exemplars. Like all papers that rely on synthetic abstractions, our goal was to develop such hypotheses to explain an interesting phenomena seen in practical scenarios. A promising future direction for our work thus is to test the hypotheses we have formulated in large language models, as well as generalize and test the mechanistic interpretation of the learned transformer algorithm in practical scenarios.

Impact Statement

This paper provides a comprehensive scientific analysis of a Transformer model that solves a small-scale synthetic task. We believe that the scientific findings presented in this paper will lay the groundwork for the development of more reliable and interpretable AI systems for the benefit of society.

References

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 1, context-free grammar. *arXiv preprint arXiv:2305.13673*, 2023.

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., and et al., S. S. Palm 2 technical report, 2023.

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Gianinazzi, L., Gajda, J., Lehmann, T., Podstawski, M., Niewiadomski, H., Nyczyk, P., et al. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*, 2023.

Broadbent, S. R. and Hammersley, J. M. Percolation processes: I. crystals and mazes. In *Mathematical proceedings of the Cambridge philosophical society*, volume 53, pp. 629–641. Cambridge University Press, 1957.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Chen, A., Phang, J., Parrish, A., Padmakumar, V., Zhao, C., Bowman, S. R., and Cho, K. Two failures of self-consistency in the multi-step reasoning of llms. *arXiv preprint arXiv:2305.14279*, 2023.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Chomsky, N. *Syntactic structures*. Mouton de Gruyter, 2002.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to algorithms*. MIT press, 2022.

Creswell, A. and Shanahan, M. Faithful reasoning using large language models. *arXiv preprint arXiv:2208.14271*, 2022.

Creswell, A., Shanahan, M., and Higgins, I. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*, 2022.

Dziri, N., Lu, X., Sclar, M., Li, X. L., Jian, L., Lin, B. Y., West, P., Bhagavatula, C., Bras, R. L., Hwang, J. D., et al. Faith and fate: Limits of transformers on compositionality. *arXiv preprint arXiv:2305.18654*, 2023.

Feng, G., Gu, Y., Zhang, B., Ye, H., He, D., and Wang, L. Towards revealing the mystery behind chain of thought: a theoretical perspective. *arXiv preprint arXiv:2305.15408*, 2023.

Gemini, T., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Hao, S., Gu, Y., Ma, H., Hong, J. J., Wang, Z., Wang, D. Z., and Hu, Z. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Karpathy, A. *NanoGPT*, 2021. Github link. https://github.com/karpathy/nanoGPT.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

LaValle, S. Planning algorithms. *Cambridge University Press google schola*, 2:3671–3678, 2006.

Li, Y., Sreenivasan, K., Giannou, A., Papailiopoulos, D., and Oymak, S. Dissecting chain-of-thought: A study on compositional in-context learning of mlps. *arXiv preprint arXiv:2305.18869*, 2023.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Lu, Y., Bartolo, M., Moore, A., Riedel, S., and Stenetorp, P. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

Lubana, E. S., Bigelow, E. J., Dick, R. P., Krueger, D., and Tanaka, H. Mechanistic mode connectivity. In *International Conference on Machine Learning*, pp. 22965–23004. PMLR, 2023.

Momennejad, I., Hasanbeig, H., Frujeri, F. V., Sharma, H., Ness, R. O., Jojic, N., Palangi, H., and Larson, J. Evaluating cognitive maps in large language models with cogeval: No emergent planning. *Advances in neural information processing systems*, 37, 2023.

Navarro, G. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

OpenAI. Gpt-4 technical report. *arXiv*, pp. 2303–08774, 2023.

Pal, A., Umapathi, L. K., and Sankarasubbu, M. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Conference on Health, Inference, and Learning*, pp. 248–260. PMLR, 2022.

Press, O. and Wolf, L. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Press, O., Zhang, M., Min, S., Schmidt, L., Smith, N. A., and Lewis, M. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

Prystawski, B. and Goodman, N. D. Why think step-by-step? reasoning emerges from the locality of experience. *arXiv preprint arXiv:2304.03843*, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ramesh, R., Khona, M., Dick, R. P., Tanaka, H., and Lubana, E. S. How capable can a transformer become? a study on synthetic, interpretable tasks. *arXiv preprint arXiv:2311.12997*, 2023.

Razeghi, Y., Logan IV, R. L., Gardner, M., and Singh, S. Impact of pretraining term frequencies on few-shot reasoning. *arXiv preprint arXiv:2202.07206*, 2022.

Saparov, A. and He, H. 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`.

Schaeffer, R., Pistunova, K., Khanna, S., Consul, S., and Koyejo, S. Invalid logic, equivalent gains: The bizarreness of reasoning in language model prompting. *arXiv preprint arXiv:2307.10573*, 2023.

Shah, H., Tamuly, K., Raghunathan, A., Jain, P., and Netrapalli, P. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33: 9573–9585, 2020.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Turpin, M., Michael, J., Perez, E., and Bowman, S. R. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *arXiv preprint arXiv:2305.04388*, 2023.

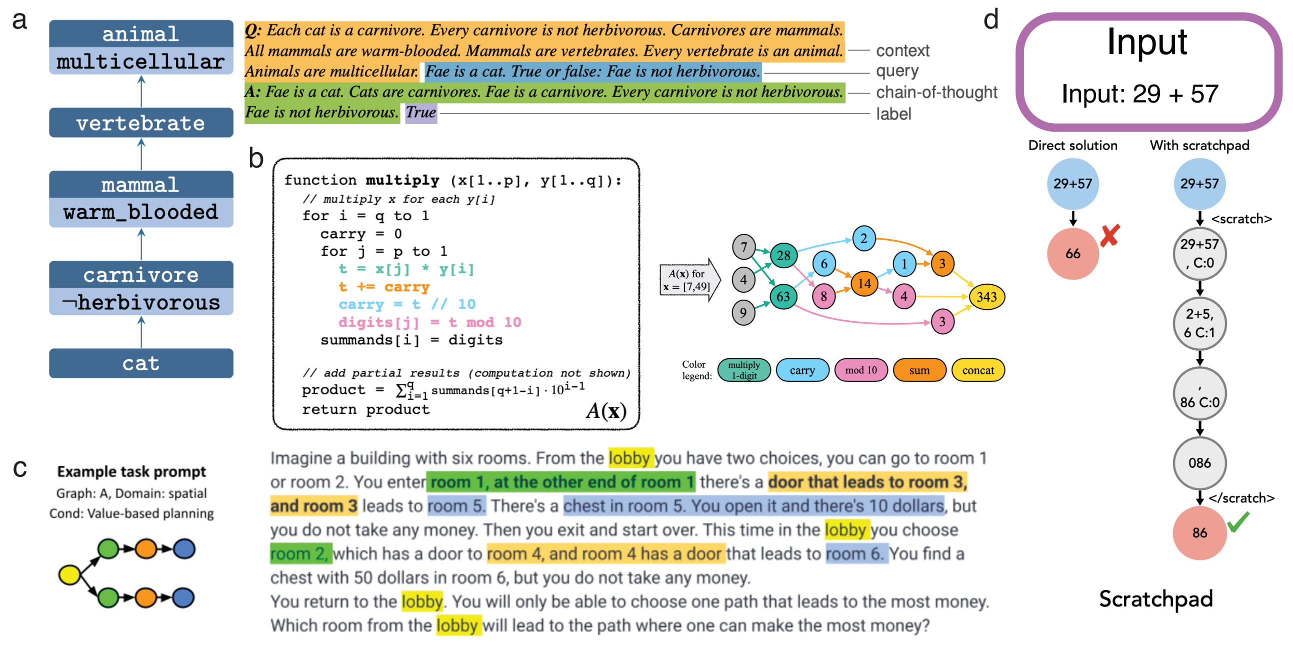
Wang, B., Min, S., Deng, X., Shen, J., Wu, Y., Zettlemoyer, L., and Sun, H. Towards understanding chain-of-thought prompting: An empirical study of what matters. *arXiv preprint arXiv:2212.10001*, 2022a.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022b.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Zelikman, E., Mu, J., Goodman, N. D., and Wu, Y. T. Star: Self-taught reasoner bootstrapping reasoning with reasoning. 2022.



Q: Each cat is a carnivore. Every carnivore is not herbivorous. Carnivores are mammals. All mammals are warm-blooded. Mammals are vertebrates. Every vertebrate is an animal. Animals are multicellular. Fae is a cat. True or false: Fae is not herbivorous.
A: Fae is a cat. Cats are carnivores. Fae is a carnivore. Every carnivore is not herbivorous. Fae is not herbivorous. True

```
function multiply (x[1..p], y[1..q]):
  // multiply x for each y[i]
  for i = q to 1
    carry = 0
    for j = p to 1
      t = x[j] * y[i]
      t += carry
      carry = t // 10
      digits[j] = t mod 10
    summands[i] = digits

  // add partial results (computation not shown)
  product = $\sum_{i=1}^{q}$ summands[q+1-i] · $10^{i-1}$
  return product
```
$A(\mathbf{x})$

Imagine a building with six rooms. From the lobby you have two choices, you can go to room 1 or room 2. You enter room 1, at the other end of room 1 there's a door that leads to room 3, and room 3 leads to room 5. There's a chest in room 5. You open it and there's 10 dollars, but you do not take any money. Then you exit and start over. This time in the lobby you choose room 2, which has a door to room 4, and room 4 has a door that leads to room 6. You find a chest with 50 dollars in room 6, but you do not take any money.
You return to the lobby. You will only be able to choose one path that leads to the most money. Which room from the lobby will lead to the path where one can make the most money?

Figure 10. **Examples of stepwise inference as graph navigation in LLM evaluations:** [Figures taken from respective papers] (a) An example graph created for a prompt (left) from the ProntoQ&A dataset (Saparov & He, 2023) (b) (Dziri et al., 2023) studies how simple algorithms such as multiplication of digits can be represented as a graph (c) CogEval (Momennejad et al., 2023) studies many large scale LLMs such as ChatGPT-4 and Claude2 on planning and navigation tasks. (d) Mathematical expression evaluation in the case of additionof two numbers can be visualized as a series of steps of a digit-wise addition algorithm.

A. Why graph navigation?

In this section we will elaborate on our paradigm of graph navigation to study stepwise inference. We first describe examples of various computational tasks that can be cast as graph navigation.

- **First order logic:** Saparov & He (2023) study simple DAGs as models of first order logical reasoning. They construct *ontologies* (see Fig. 10a) and prompt LLMs to do analogical reasoning.

- **Mathematical expression evaluation:** Dziri et al. (2023) study mathematical expression evaluation in large scale LLMs as DAG navigation (see Fig. 10b). Any mathematical expression can be decomposed into elementary computations which are chained together.

- **Planning and spatial navigation:** Momennejad et al. (2023) evaluates many large scale LLMs such as ChatGPT-4 and Claude2 on synthetically designed planning and navigation tasks (see Fig. 10c).

- **Formal grammars and natural language:** Allen-Zhu & Li (2023) studies transformers trained on context-free grammars (CFGs) which are DAGs. Another motivation for the study of graph navigation comes from linguistics and natural language syntax (Chomsky, 2002). Every sentence in a language can broken down into its syntactic or parse tree, a special case of a directed acyclic graph. For example, the sentence 'I drive a car to my college' can be parsed as the following graph: ('I': Noun phrase, 'drive a car to my college': Verb Phrase) → ('drive': Verb, 'a car': Noun Phrase, 'to my college': Prepositional Phrase) → ('a': Determiner, 'car': Noun), ('to': Preposition, 'my college': Noun Phrase) → ('my': Determiner, 'college': Noun).

Effective stepwise reasoning consists of several elementary logical steps put together in a goal-directed path that terminates at a precise state (LaValle, 2006). We argue that graph navigation problems provide such a fundamental framework for studying stepwise inference. Graphs provide a universal language for modeling and solving complex problems across various domains. Whether it is optimizing network traffic, analyzing social networks, sequencing genetic data, or solving puzzles like the Travelling Salesman Problem, the underlying structure can often be mapped onto a graph (Cormen et al., 2022; Momennejad et al., 2023; Dziri et al., 2023; Saparov & He, 2023).

B. Detailed Related Work

Large language models (LLMs) have been shown to possess sophisticated and human-like reasoning and problem-solving abilities (Srivastava et al., 2022). Chain-of-thought or scratchpad reasoning refers to many similar and related phenomena involving multiple intermediate steps of reasoning *generated internally and autoregressively* by the language model. First described by Nye et al. (2021); Kojima et al. (2022), adding prompts such as '`think step by step`' allows the LLM to autonomously generate intermediate steps of reasoning and computation, improving accuracy and quality of its responses. This is referred to as zero-shot chain-of-thought. A related set of phenomena, few-shot chain-of-thought prompting (Wei et al., 2022) occurs when the language model is shown exemplars of reasoning before being prompted with a reasoning task. The model follows the structure of logic in these exemplars, solving the task with higher accuracy. Further, there have been several prompting strategies developed, all of which rely on sampling intermediate steps: tree-of-thoughts (Yao et al., 2023), graph-of-thoughts (Besta et al., 2023), program-of-thoughts (Chen et al., 2022), self-ask (Press et al., 2022) and methods which use more than 1 LLM: such as STaR (Zelikman et al., 2022), RAP (Hao et al., 2023), Selection-Inference (SI) (Creswell et al., 2022; Creswell & Shanahan, 2022). Dziri et al. (2023) study how LLMs solve multi-step reasoning tasks and argue that models *likely* fail because they reduce most multi-step reasoning tasks to linearized sub-graph matching, essentially learning 'shortcut solutions' (Liu et al., 2022). Momennejad et al. (2023) study in-context graph navigation in LLMs, finding that they fail to do precise planning. Recently, a few works have used theoretical approaches to characterize and explain stepwise inference. Li et al. (2023) study in-context learning of random MLPs and finds that a transformer that outputs the values of intermediate hidden layers achieves better generalization, Feng et al. (2023) shows that with stepwise reasoning, transformers can solve dynamic programming problems, and Prystawski & Goodman (2023) studies reasoning traces in transformers trained to learn the conditionals of a Bayes network. There are several puzzling phenomena in the prompts used to elicit few-shot chain-of-thought reasoning: chain-of-thought can be improved by sampling methods such as self-consistency (Wang et al., 2022b), prompts might not reflect the true reasoning process used by the language model, as identified by Turpin et al. (2023), and the accuracy of the model can be sensitive to the order in which prompts are provided (Lu et al., 2021). Saparov & He (2023) introduce a synthetic dataset called PrOntoQA to systematically study the failure modes of chain of thought in the GPT3 family fine-tuned on the dataset and find that misleading steps of reasoning are a common cause of failure in the best-performing models. Chen et al. (2023) find that chain-of-thought fails at compositional generalization and counterfactual reasoning. Wang et al. (2022a); Schaeffer et al. (2023) find that the content of the exemplars is less relevant to accuracy than their syntactic structure. Razeghi et al. (2022) find that the accuracy of reasoning is correlated with the frequencies of occurrence in the pretraining dataset.

C. Setup and construction of graph and model

C.1. Graph structures

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

We use two DAG structures, hierarchical and Bernoulli (Fig. 11). Bernoulli 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. Below, we provide algorithms to generate our graph structures.

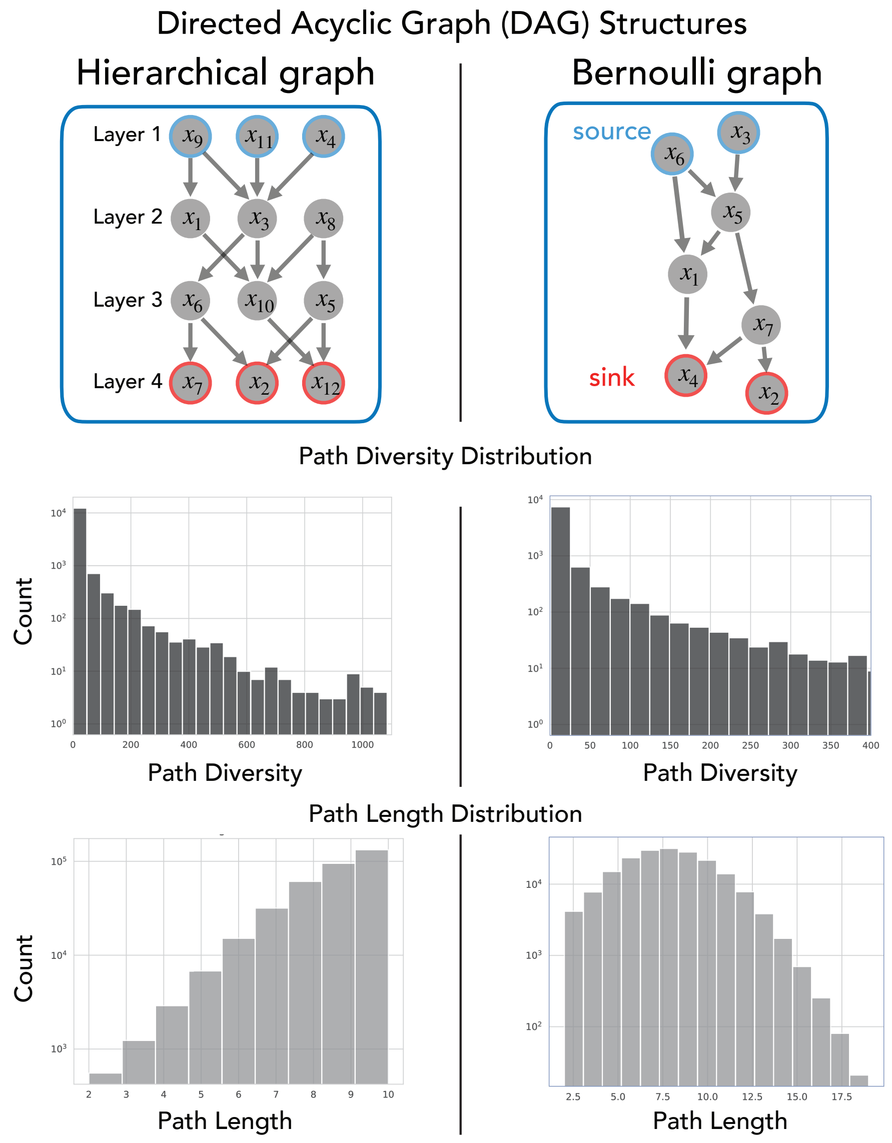


Figure 11. **Construction and properties of Hierarchical and Bernoulli DAGs:** (top) Schematic of hierarchical and Bernoulli graphs. Hierarchical graphs are organized into layers with connections only between nodes of successive layers but Bernoulli graphs have no such structure. (middle) Path diversity is defined as the number of paths between any 2 path connected nodes. (bottom) Path length distributions: Owing to the hierarchical nature, the path length distribution is exponential in hierarchical graphs whereas it is more Gaussian-like for Bernoulli graphs.

Algorithm 1 Generate Bernoulli connected DAG

```
Require: $numNodes > 0$, probability $p$ for edges
 1: nodeNames ← ['X' + str(i) for i in range(numNodes)]
 2: Function CreateUpperTriangularMask(n, p)
 3: matrix ← random binary matrix with size $n \times n$ and probability $p$ for 1s
 4: upperTriangular ← extract upper triangular part of matrix
 5: return upperTriangular
 6: repeat
 7:    adjMatrix ← CreateUpperTriangularMask(numNodes, p)
 8:    dag ← create directed graph in NetworkX from adjMatrix and nodeNames
 9: until dag is connected
```

Algorithm 2 Generate Hierarchical Connected DAG

```
1:  p ← [probability of connection between layers]
2:  nodesPerLayer ← [number of nodes in each layer]
3:  numLayers ← [total number of layers]
4:  numNodes ← nodesPerLayer × numLayers
5:  Function CreateLayeredDAG(nodesPerLayer, numLayers, p)
6:  Initialize an empty directed graph G in NetworkX
7:  for currentLayer ← 1 to numLayers − 1 do
8:      for each node $j$ in currentLayer do
9:          for each node $k$ in currentLayer + 1 do
10:             if random number ≤ p then
11:                 Add edge from node $X_j$ to node $X_k$ in G
12:             end if
13:         end for
14:     end for
15: end for
16: return G
17: End Function
18: repeat
19:     dag ← CreateLayeredDAG(nodesPerLayer, numLayers, p)
20: until dag is connected
```

C.2. Motif construction

In the multi-graph scenario, we first construct a set of n graphs (in our experiments, we use Bernoulli DAGs with $n = 10$) denoted by $G = \{g_1, g_2, ..., g_n\}$. To construct the train data, we first create all pairwise motif orders $\{(g_i \to g_j)\}$ and leave some of them for evaluation. For our experiments, we held out 10 out of 45 motif orders.

C.2.1. Construction of exemplar sequences

To provide examples in-context, we create exemplar sequences connecting motifs, say g_{i_1} and g_{i_2}. In our construction, we select X_s to be *source node* in g_{i_1} and X_g to be a *sink node* in g_{i_2}. Further, we choose a *sink* of g_{i_1}, $X_{\text{sink}}(g_{i_1})$ and a *source* of g_{i_2}, $X_{\text{source}}(g_{i_2})$ and connect them via a ghost edge: $(X_{\text{sink}}(g_{i_1}), X_{\text{source}}(g_{i_2}))$. These intermediate nodes are **subgoals** for the path that the model has to produce. Finally putting everything together, the exemplar sequence has the following form: goal: X_g $X_s \ldots X_{\text{sink}}\,(g_{i_1}) X_{\text{source}}(g_{i_2}) \ldots X_g$. Here, $X_s \ldots X_{\text{sink}}$ is a path from a source to a sink in g_{i_1} and $X_{\text{source}}(g_{i_2}) \ldots X_g$ is a path from a source to a sink in g_{i_2}. To be precise, we summarize this process into the algorithm below:

Algorithm 3 Generate In-context Exemplars

```
Require: $\{g_{i_1}, g_{i_2}\}$, 2 motifs across which a ghost edge will be placed.
1: $X_s$ ← Sample sources($g_{i_1}$)
2: $X_g$ ← Sample sinks($g_{i_2}$)
3: $(X_{\text{pre}}^{\text{ghost edge}}, X_{\text{post}}^{\text{ghost edge}})$ ← (Sample sinks($g_{i_1}$), Sample sources($g_{i_2}$))
4: $(X_s \ldots X_{\text{pre}}^{\text{ghost edge}})$ ← Sample path $g_{i_1}$
5: $(X_{\text{post}}^{\text{ghost edge}} \ldots X_g)$ ← Sample path $g_{i_2}$
6: return $e_{g_{i_1}, g_{i_2}}$ ← $X_s \ldots X_{\text{pre}}^{\text{ghost edge}} X_{\text{post}}^{\text{ghost edge}} \ldots X_g$
```

Further, after providing a set of exemplar sequences in-context, we chain them together to create a longer sequence. To be precise, given a set of K motifs $\{g_{i_1}, g_{i_2}, g_{i_3}, \ldots g_{i_K}\}$, we have the set of K-1 ghost edges, one for each exemplar: $\{(X_{\text{sink}}(g_{i_1}), X_{\text{source}}(g_{i_2})), (X_{\text{sink}}(g_{i_2}), X_{\text{source}}(g_{i_3})), \ldots (X_{\text{sink}}(g_{i_{K-1}}), X_{\text{source}}(g_{i_K}))\}$. To create the final path, we choose goal $X_g \in g_{i_1}$ and start $X_s \in g_{i_K}$. This path has every ghost edge from the list present in it.

C.3. Architecture details and loss function

Loss function

For training, we tokenize every node and we use the standard language modeling objective, next-token prediction with a cross entropy loss. Here target$_n$ is the 1-shifted version of the training sequence and $\mathbf{x}_n$ are the logit outputs of the model at the n^{th} timestep.

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

Hyperparameter	Value
learning rate	10^{-4}
Batch size	64
Context length	32
Optimizer	Adam
Momentum	0.9, 0.95
Activation function	GeLU
Number of blocks	2
Embedding dimension	64

Table 1. Hyperparameters of the transformer models used for all experiments except mechanistic analyses

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

The transformer architecture consists of repeated blocks of pre-LayerNorm, causal multi-head self-attention, post-LayerNorm, and an MLP with skip connections (see Fig. 12). The MLP contains two fully-connected layers with a GELU non-

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

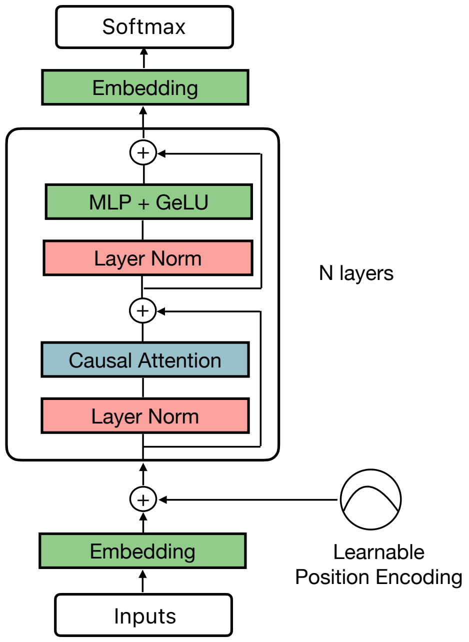


Figure 12. **The architecture of GPT-style (Radford et al., 2019) decode-only transformers.** Note the presence of both pre and post-LayerNorm in each transformer block. Figure from methods section of Ramesh et al. (2023).

linearity (Hendrycks & Gimpel, 2016). The dimensionality of the hidden layer of the MLP is 4x the embedding dimensionality. We do not include any dropout in the model or biases in the linear layers. We use weight-tying (Press & Wolf, 2016) in the embedding and un-embedding layers.

To do the mechanistic study, we consider a 1 layer attention-only transformer without a few modifications: We remove the MLP and post-LayerNorm and vary the embedding dimensionality from 4 to 64. This 1L transformer is described by the following model equations. Here $\mathbf{X}_{\text{token}} \in \mathcal{R}^{\text{vocab size} \times \text{T}}$ denotes the tokens, $W_E \in \mathcal{R}^{n_{\text{embd}} \times \text{vocab size}}$ is the positional embedding matrix, $W_{\text{pos}} \in \mathcal{R}^{n_{\text{embd}} \times T}$ is the token embedding matrix and $\mathbf{X} \in \mathcal{R}^{n_{\text{embd}} \times \text{T}}$:

$$\begin{aligned}
\mathbf{X} &= W_E(\mathbf{X}_{\textbf{token}}) + W_{\text{pos}}(\mathbf{X}_{\textbf{token}}) \\
\mathbf{X} &= \text{LN}(\mathbf{X}) \\
\mathbf{X} &= \mathbf{X} + \text{softmax}(\mathbf{X}^T W_Q^T W_K \mathbf{X}) W_V \mathbf{X} \\
\mathbf{z} &= \text{softmax}(W_E^T \mathbf{X}) \\
\text{next token} &= \text{argmax}_{\text{all tokens}} \mathbf{z}
\end{aligned}$$

D. Training protocol for experiments

For experiments in our setup without exemplars, we randomly generate either a hierarchical graph or a Bernoulli graph G with $N = 200$ nodes. In the Bernoulli graph setting the probability of an edge $p = 0.05$, while in the hierarchical graph, the probability of an edge between a node in layer l and layer $l + 1$ is $p = 0.05$. We choose 10 layers with 20 nodes each to match the number of nodes in the two graph types. We convert all the nodes to tokens, along with a special `goal` token which corresponds to a [`BOS`] token and an `end` token which corresponds to an [`EOS`] token and another token for padding, `pad`.

Test-train split: To generate training data corresponding to path connected node pairs, we first put all edges (which are paths of length 1) into the training data. This procedure was done in all experiments to ensure that full knowledge of the graph was presented to the model. Further, we generate all simple paths between every pair of nodes in G. A variable fraction of these paths are included in the training data, depending on experimental conditions which we outline below.

For experiments in Figs. 3a-b, we pick 20% of the path-connected nodes and put all simple paths between them into the training data, for each graph type. We also add an equal number of non-path connected nodes to the training data.

In Fig. 3c, for each value of the path-length threshold parameter, which sets the maximum length of paths in the training dataset, we pick paths corresponding to 20% of the allowed path-connected node pairs and put them into the training data, while the remainder are held out evaluations. For the non-path connected pairs, we simply take all node pairs that are not path-connected and add a fraction of these node pairs into the training data, chosen to roughly balance the number of path-connected node pairs according to the experimental conditions. The rest are held-out for evaluation.

For the motif experiments in Fig. 8, we generate a set of 10 motifs, each with a Bernoulli graph structure of 100 nodes with a bernoulli parameter $p = 0.95$. We then divide the 45 possible motif orders into a set of 35 and 10 that we put into train and test respectively. For generating the context, we combine 3-6 motifs according to the allowed orders, and then sample exemplars as well as the final sequence that traverses the full motif chain by choosing start and goal nodes from the set of sources and sinks respectively.

E. Additional experimental results

Label noise in training data In Fig. 13, we mimic real-world language data, abundant in ambiguity and polysemy, by corrupting (a) 5%, (b) 10% and (c) 20% of tokens in a single graph scenario. To achieve this, we replaced a randomly chosen 5% and 10% of the tokens in the training data with random tokens. We observe that the gap between stepwise inference and direct inference persists in both scenarios. This finding indicates that stepwise inference remains effective in more realistic settings with noise.

Varying edge density In Fig. 14, 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. This is because

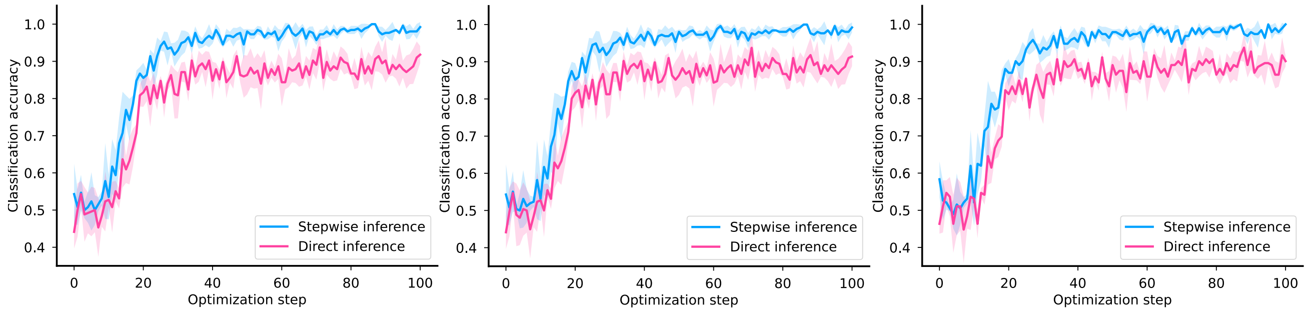


Figure 13. **Persistence of stepwise inference gap with corrupted tokens:** In this experiment with setup identical to Fig. 3a-b, (a) 5%, (b) 10% and (c) 20% of tokens were randomly corrupted to mimic real world language data. The stepwise-inference gap persists.

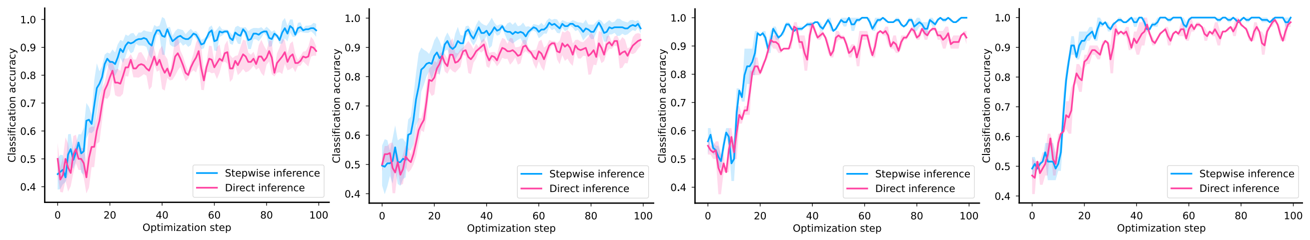


Figure 14. **The effect of varying edge density in the single graph scenario:** Here we vary p, the edge density of connectivity in the graph, from 0.08 in the left-most plot to 0.12 in the right-most plot, in steps of 0.01. The stepwise inference gap persists in all cases.

the more likely the nodes are to be connected, the more likely it is for shortest paths to exist between nodes and thus less "stitching" is needed (Broadbent & Hammersley, 1957).

Short path bias Fig. 15 presents a density plot comparing the average lengths of actual paths with those generated by the model in a Bernoulli graph. This observation verifies the model tends to produce shorter paths between a given pair of start and goal nodes.

Effect of varying embedding dimensionality in the single graph scenario Here we consider the 1-layer Transformer without MLP and post-LayerNorm and ask the following question: For a fixed underlying graph size and training data, how does the model performance vary as we sweep embedding dimensionality. Intuitively, the higher the embedding dimensionality is, the model should be able to generalize better by learning a better embedding of the node tokens. We see that beyond a critical dimensionality (which is around 20 for a graph of size 200 nodes), the model generalizes to all held out (start, goal) node pairs with a fairly abrupt transition, Fig. 16.

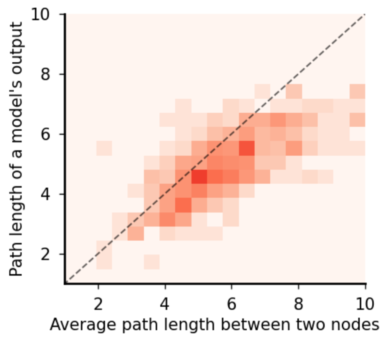


Figure 15. **Model outputs are biased toward shorter paths.**

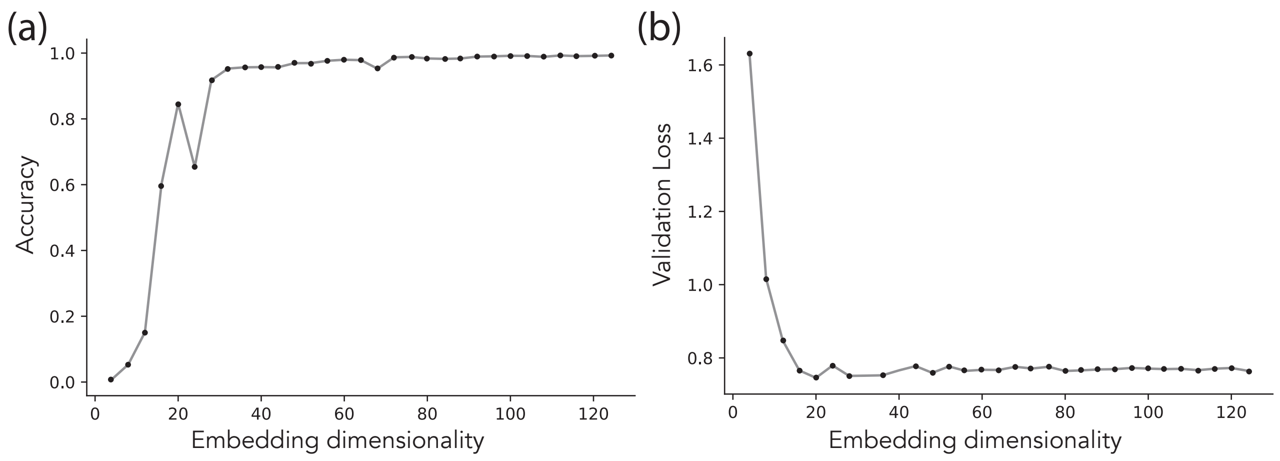


Figure 16. **Varying embedding dimensionality in 1 layer models:** We find that there is a critical embedding dimensionality (around 20 for a Bernoulli graph of size 200 nodes and $p = 0.05$) above which the model can generalize to all held-out node pairs.

F. Intuition guiding the mechanistic analysis

In this section, we present the intuition that served as the hypothesis guiding our mechanistic analysis.

Consider the optimal maximum likelihood estimator designed to solve our graph navigation task. Given a start node X_s and an incomplete sequence of predicted nodes $X_1, \dots, X_k$ in the pursuit of navigating to the goal node X_g, the estimator works the following way:

$$X_{\text{next}} = \arg\max_{X'} P(X'|X_s; X_1, \dots, X_k; X_g)$$

Since the task is conditionally Markovian: the choice of the next step will be independent of the history when conditioned on X_g and X_k. Accordingly, we have:

$$\begin{aligned} X_{\text{next}} &= \text{argmax}_{X'} P(X'|X_k, X_g) \\ &= \arg\max_{X'} \frac{P(X_g|X', X_k)P(X', X_k)}{P(X_g, X_k)} \end{aligned}$$

This decomposition leads to interpretable terms which shed light on what algorithm the model might use:

$$\log P(X'|X_k, X_g) = \log P(X_g|X', X_k) + \log P(X', X_k) - \log P(X_g, X_k)$$

These terms can be interpreted as follows:

1. $\log P(X_g, X_k)$ describes the prompt.
2. $\log P(X', X_k)$ describes the knowledge of the **world model**: How well does the model know the ground truth structure of the graph?
3. $\log P(X_g|X', X_k)$ corresponds to **goal-directed behavior**: What X' is most likely to lead to the goal?

Let $\mathcal{C}(X_k)$ denote the subset of nodes in the graph that are *children* of the node X_k. Then, while selecting the next token that has the highest likelihood, note that terms (1) and (2) cannot be optimized over: the former does not depend on X' and the latter, for the optimal predictor, will be $1/|\mathcal{C}(X_k)|$ if $X' \in \mathcal{C}(X_k)$ and 0 otherwise. Accordingly, the only term that can be optimized over is the third one, i.e., the one that measures how likely the goal is if the next state is X'. However, due to term (2), $X' \in \mathcal{C}(X_k)$—that is, the possible set of next tokens is constrained to the set $\mathcal{C}(X_k)$.

Now, exploiting the task's conditional Markovian nature again, we have $P(X_g|X', X_k) = P(X_g|X' : X' \in \mathcal{C}(X_k))$. Heuristically, assume that $P(X_g|X') \propto \mathrm{e}^{-d(X_g, X')} \cdot \mathbb{I}(X' \in \mathcal{C}(X_k))$, where $d(X_i, X_j)$ is a measure that describes the

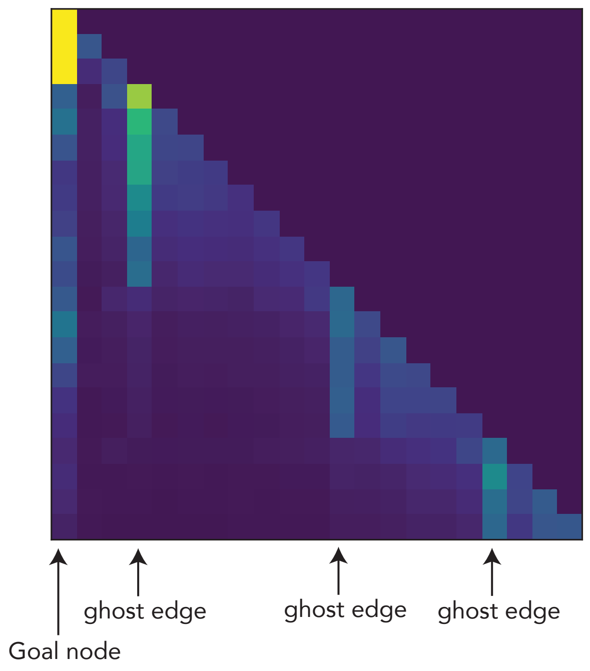


Figure 17. **Attention pattern after in-context exemplars:** Here we visualize the portion of the attention map after prompting with 4 in-context exemplar sequences. The model generates attentional patterns that treat the ghost edges as subgoals.

distance between nodes X_i and X_j, while respecting the topology of the graph, and $\mathbb{I}$ is an indicator function that is 1 if its input is True, and 0 if not. Then, we have $\log P(X_g|X', X_k) \propto -d(X_g, X') \cdot \mathbb{I}(X' \in \mathcal{C}(X_k))$.

The intuitive argument above, though likely to be approximate, suggests that a possible solution the model can learn via autoregressive training in our graph navigation setup is (i) compute the distance between all neighboring nodes of the current node and the goal node, (ii) move to the node that has the least distance, and (iii) repeat. The algorithm we uncover in our analysis in Sec. 3.1.4 in fact functions in a similar way: the model is constantly computing a inner product between the goal token's representation and the embeddings of all tokens; we find this inner product is highest for the neighbors of the current token. Then, the highest inner product token is outputted and the process is repeated. Since the embeddings are not normalized, this inner product is not exactly the Euclidean distance—we expected as much, since the topology of the graph will have to be accounted for and learning an inner product based metric will be easier for a model (because most operations therein are inner products).

Further, in the case of motifs, we expect that the model contructs a path through checkpoints defined by ghost edges, which act as subgoals. To explain, given a set of K motifs strung together in-context $\{g_{i_1}, g_{i_2}, g_{i_3}, \dots g_{i_K}\}$, we have the set of K-1 ghost edges, one for each exemplar: $\{(X_{\text{sink}}(g_{i_1}), X_{\text{source}}(g_{i_2})), (X_{\text{sink}}(g_{i_2}), X_{\text{source}}(g_{i_3})), \dots (X_{\text{sink}}(g_{i_{K-1}}), X_{\text{source}}(g_{i_K}))\}$. Thus, we hypothesize that the model identifies all K-1 ghost edges from its context and plans sub-paths to each ghost edge in pursuit of the goal. Preliminary analyses of attention patterns Fig. 17 provides evidence for this hypothesis.

F.1. Generalizing static word embeddings to 3-way relations

Static word embedding algorithms such as Word2vec are trained by sampling pairs of words (w_i, w_c) that appear in the same context and adjusting their embeddings so that their inner product is higher than an inner product with the embedding of word w_i and a randomly sampled word from the vocabulary. The algorithm can be understood as performing a low-rank factorization of the matrix of co-occurrence statistics. In the case of Word2vec, the matrix is factorized as $P = I \cdot C$ where I contains *word vectors* in its rows and C contains *word covectors* in its columns. Therefore, every word has two types of embeddings. One is used when the word appears in the first position in the pair (which corresponds to center words), and the second when it appears in the second position (which corresponds to context words).

Inspired by the interpretability results, we propose to solve the graph navigation task using a similar low-rank factorization method but generalized to 3-way relations. In this case, the tensor T to be factorized is third-order, and for each node, we have three types of embeddings: One, which is used when the node acts as a goal, one when it acts as the current state, and one when it acts as the next possible state.

We do not deal with a natural corpus with different frequency of occurrence of individual nodes, and therefore we set the numbers T_{ijk} to be proportional to the length l_{ijk} of a path which goes through an ordered pair of neighbour nodes (i, j) to a node k. If there is no such path, we set the length to ∞. The target value T_{ijk} can be seen as a preference for a node j when the goal is to reach the node k from the node i and it is equal to $l_{ijk}^{-1}/\sum_{j'} l_{ij'k}^{-1}$.

Inspired by the learned algorithm, we can use low-rank tensor factorization to approximate this matrix by the following expression $T_{ijk} \approx \hat{T}_{ijk} = (\mathbf{s}_i + \mathbf{g}_k) \cdot \mathbf{n_i}$ where $\mathbf{s_i}$, $\mathbf{g}_k$, $\mathbf{n_i}$ are the three types of learnable embeddings. Therefore, by interpreting the trained transformer, we can obtain a simple algorithm that can be potentially useful in setups that deal with 3-way relationships. We leave this for future work.