SPECTRAL JOURNEY: HOW TRANSFORMERS PREDICT THE SHORTEST PATH

Andrew Cohen, Andrey Gromov, Kaiyu Yang, Yuandong Tian
Meta, FAIR
{andrewcohen,gromovand,kaiyuy,yuandong}@meta.com

ABSTRACT

Decoder-only transformers lead to a step-change in capability of large language models. However, opinions are mixed as to whether they are really planning or reasoning. A path to making progress in this direction is to study the model's behavior in a setting with carefully controlled data. Then interpret the learned representations and reverse-engineer the computation performed internally. We study decoder-only transformer language models trained from scratch to predict shortest paths on simple, connected and undirected graphs. In this setting the representations and the algorithm learned by the model are completely interpretable. We present three major results: (1) Two-layer decoder-only language models can learn to predict shortest paths on simple, connected graphs containing up to 10 nodes. (2) Models learn a graph embedding that is correlated with the spectral decomposition of the *line graph*. (3) A new, approximate path-finding algorithm Spectral Line Navigation that relies on the spectral decomposition of the line graph to compute shortest paths.

1 INTRODUCTION

The rise of decoder-only transformers led to a dramatic change in the capabilities and adoption of large language models (Vaswani et al., 2017; Achiam et al., 2023; Anthropic, 2024; Touvron et al., 2023; Dubey et al., 2024). Despite the apparent success, it remains unclear whether the models are capable of *planning* or *reasoning* (Kambhampati et al., 2024; Mirzadeh et al., 2024). The lack of clarity is partially due to absence of a quantitative, verifiable definition of planning or reasoning. For this reason, language models are typically evaluated via performance on benchmarks focusing on math problems or multi-hop composition, where finding solutions requires humans to reason (Cobbe et al., 2021; Yang et al., 2024). Unfortunately, these results cannot distinguish "true reasoning/planning" from pattern matching or retrieval due to web-scale training and possible data contamination. To make matters worse, models are often very sensitive to the exact prompt, leading to brittleness when evaluated on variations of math problems with changed variable values or irrelevant clauses (Mirzadeh et al., 2024). In other work, brittleness is shown to be correlated with test set contamination (Oren et al., 2023).

One way to progress is to go beyond benchmark numbers and study language model *internal dynamics* in settings with carefully controlled, structured data (Ye et al., 2024; Edelman, 2024). In such settings, it may be possible to define the notion of reasoning precisely, by showing that model learned a general algorithm from a few examples. This algorithm can be extracted by interpreting the representations developed by the model. Mechanistic interpretability (MI) is a research area that aims to understand representations and reverse-engineer the computations learned by neural networks (Elhage et al., 2021). It has revealed that models can learn general algorithms and representations in algorithmic tasks such as modular arithmetic (Power et al., 2022; Nanda et al., 2023; Gromov, 2023; Liu et al., 2022a; He et al., 2024), path-finding in directed acyclic graphs (Khona et al., 2024), Markov chains (Nichani et al., 2024), iterative algorithms (Cabannes et al., 2024) and generating complex objects in diffusion models (Okawa et al., 2024). Some MI work has proved useful for language models (Templeton et al., 2024) and general ML tasks (Liu et al., 2022b).

We argue that investigating language model behavior (during training *and* inference) in a general abstract problem space that requires reasoning, such as graphs and graph operations, may lead to a

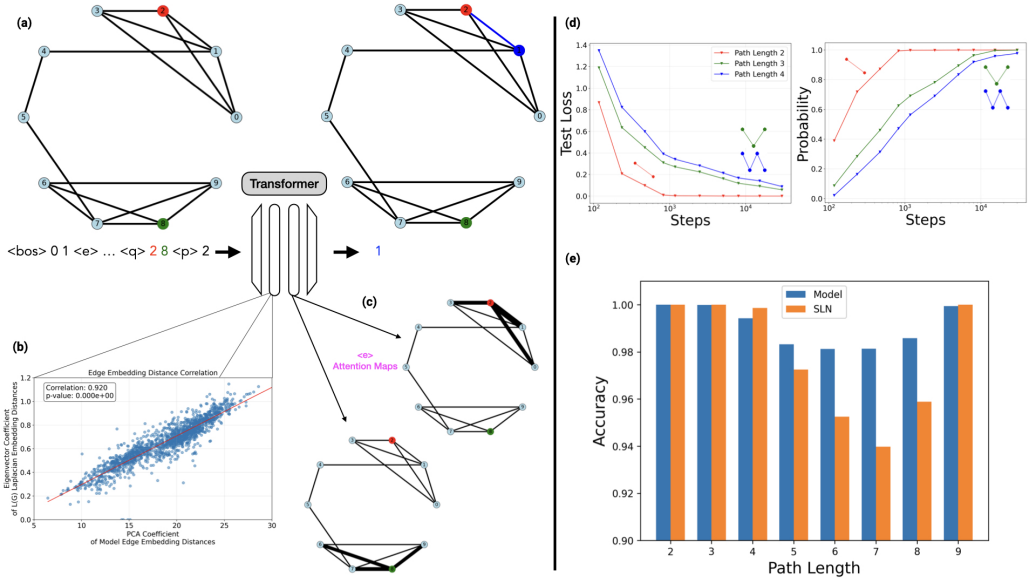

Figure 1: *Overview* **(a)** We train 2-layer transformers to predict nodes in the *shortest path* between a source and target node for a given graph represented sequentially as a list of edges and nodes. **(b)** We find a strong correlation between model embeddings in layer 1 and the spectral decomposition of the graph and **(c)** attention head dynamics (attention activations denoted by thickness of edge) in layer 2 which attend to the the current and target node edge tokens. Using this, we derive, implement, and evaluate a novel (approximate) path-finding algorithm Spectral Line Navigation (*SLN*). **(d)** During training, the model first learns to predict paths with 2 nodes (connected by a single edge) and then learns an algorithm for paths with >2 nodes. Accuracy on paths of length 3, 4 and beyond improve simultaneously. **(e)** After training, the model achieves 99.42% accuracy on the test set and *SLN* achieves 99.32% accuracy.

taxonomy of algorithms and computations learned by language models for reasoning tasks. Graphs can model many reasoning and planning problems, such as Trivia, navigation, and math word problems (). Additionally, the wealth of research on graph theory in computer science and mathematics provides many useful ideas for analysis.

In this work, we study GPT-style transformer models trained from scratch to predict shortest paths on simple, connected, and undirected graphs. Then, we inspect the attention maps and learned representations to identify the algorithm employed by the model to compute the shortest path. Surprisingly, we find that two-layer models can learn to perform this task on graphs with up to 10 nodes, while one-layer models cannot. Mechanistic analysis shows *no evidence* of traditional dynamic programming approaches (Dijkstra, 1959; Bellman, 1960; Brinkmann et al., 2024). Instead, we find evidence of an algorithm that is *spectral* in nature (Chung, 1997). At a high level, the model learns an embedding scheme of edges which is correlated with their distance in the graph and then selects nodes using the minimum of this distance. We then implement this algorithm directly (without relying on the neural network) and evaluate it on the test set (described in the following sections) getting 99.32% accuracy. We refer to this spectral algorithm as Spectral Line Navigation (*SLN*). Although spectral methods for finding shortest paths have been (briefly and recently) studied in the graph theory literature (Steinerberger, 2020), to the best of our knowledge, *SLN* is a novel (approximate) shortest path finding algorithm. Similar algorithmic ideas have been recently discovered in previous MI work (Khona et al., 2024).

The major results of this work are as follows:

- Two-layer decoder-only transformer models can learn to predict shortest paths on simple, connected graphs containing up to 10 nodes. Furthermore, models with a single attention head can learn the task, but increasing the headcount while keeping the number of parameters fixed allows the model to learn the task faster.

- Models learn a graph embedding that is correlated with the spectral decomposition of the line graph[1]. Furthermore, models strongly attend to the edges containing the current node and the target node when selecting a next edge in the path.
- An approximate path-finding algorithm *SLN*, which uses the distance between edges in the spectral decomposition of the line graph to compute shortest paths.

2 PRELIMINARIES

In this work, we study simple, connected and undirected graphs. We do not make any restrictions for cycles. A graph G is defined as $G = (V, E)$ where V is a set of nodes and E is a set of edges $E = \{(v_i, v_j) \mid v_i, v_j \in V,\ i \neq j\}$. We denote an edge between nodes v_i, v_j as $e_{i,j}$.

Shortest Path Problem Since we study connected graphs, for a source node v_{src} and target node v_{tgt}, $v_{src} \neq v_{tgt}$, there exists a sequence of nodes or *path* which connects them $P = (v_{src}, v_i, v_{i+1}, ..., v_j, v_{tgt})$ where $e_{src,i},\ e_{i,i+1},\ e_{j,tgt} \in E$. We refer to the pair (v_{src}, v_{tgt}) as the *query* and we define the length ℓ of path P as the number of nodes in P.

The shortest path problem is to find the path with the fewest nodes between v_{src} and v_{tgt}. We define the shortest path as P^* and the length of the shortest path ℓ^*. Note, P^* may not be unique. Computing shortest paths is a well-studied problem in graph theory and computer science and many algorithms exist (Cherkassky et al., 1996). In general, finding shortest paths is a challenging optimization problem as there may be many paths connecting two nodes. Selecting between these paths for the shortest requires non-trivial planning and reasoning.

Line Graph The Line Graph of a graph is another graph $L(G) = (V_L, E_L)$ which represents the adjacencies between edges (Harary & Norman, 1960). Each edge in G is represented by a node in $L(G)$ and two nodes are connected in $L(G)$ if the edges share a common node in G. Formally, $L(G)$ is constructed as:

- $V_L = \{v_{ij} \mid e_{ij} \in E\}$
- $E_L = \{e_{ijk} \mid e_{ij}, e_{jk} \in E\}$

Graph Laplacian and Spectral Decomposition The Laplacian L of a graph $G = (V, E)$ is a matrix representation defined as

$$L = D - A$$

where D is the diagonal degree matrix of nodes in G and A is the adjacency matrix of G. Nodes with high degree will have a large impact on the spectrum of L so it is common to consider the *normalized* Laplacian $\bar{L}$ defined as

$$\bar{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$

$$D^{-\frac{1}{2}}_{i,j} = \begin{cases} \frac{1}{\sqrt{deg(v_i)}} & i = j \\ 0 & i \neq j \end{cases}$$

where $deg(v_i)$ is the degree of node v_i. Then,

$$\bar{L}_{i,j} = \begin{cases} 1 & i = j \\ \frac{-1}{\sqrt{deg(v_i)deg(v_j)}} & i \neq j,\ e_{ij} \in E \\ 0 & else \end{cases}$$

The spectral decomposition of $\bar{L}$ (and L) and is a standard method for quantifying node connectivity for such tasks as finding node clusters. For connected undirected graphs, the eigenvalues are always positive and real-valued. The smallest eigenvalue is always zero and the eigenvector coefficient corresponding to the second smallest eigenvalue (i.e., the *Fiedler Vector*) corresponds to the sparsest clustering of nodes. For more fine-grained clustering, the eigenvector coefficients for the k smallest non-zero eigenvalues is used (Chung, 1997).

[1] A *line graph* is a graph obtained from a given graph by replacing all edges by nodes and adding edges if the corresponding edges in the original graph share a node

3 EXPERIMENTAL SETTINGS

Data We generate simple connected graphs of 3–10 nodes using the Labelled Enumeration algorithm (Mckay, 1983; Mckay). which yields approximately 12M non-isomorphic graphs in total. In this setting, the shortest path P^* has between 2 and 10 nodes, and thus the length $2 \leq \ell^* \leq 10$. We partition the graphs into 80% for training and 20% for testing. For each pair of nodes within a graph, we compute the shortest path for the forward and reverse directions using the Python package NetworkX (Hagberg et al., 2008).

Given a graph and query (a pair of nodes), we randomly select either the forward or reverse shortest path, but not both, to prevent the model from learning symmetry shortcuts. Although there may be multiple shortest paths between two nodes, for each sample we select only one. Samples are bucketed by both path length and the number of nodes in the graph. We then sample as uniformly as possible from these buckets given that the number of paths of a particular path length decreases as the length increases (e.g., there are fewer paths of length 9 in graphs up to 10 nodes than paths of length 4). In this manner, we generate approximately 1M training samples and 500K test samples where the test set contains exclusively graphs not in the training set.

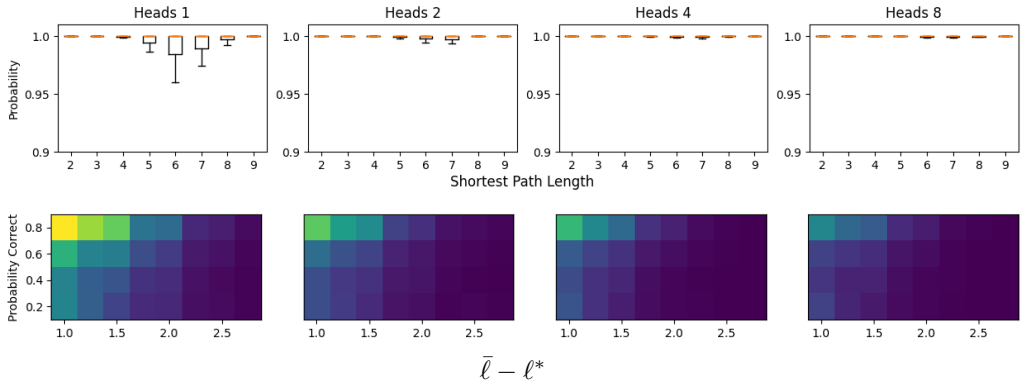

Figure 2: (**Top**) Probability of generating a shortest path by path length for 2-layer models with 1,2,4,8 heads on the test set. Increasing the number of heads improves performance, although all models are able to perform the task with high accuracy. The worst category is the 1-head model on paths of length 6 where the 1.5 interquartile range is above 0.95. (**Bottom**) The occurrence of samples by probability of correctness and the $\bar{\ell} - \ell^*$ (defined in Equation 1). Yellow is larger. Samples in which there are many paths of similar length between source and target ($\bar{\ell} - \ell^* \to 1$) contribute to the failures. As the number of heads increase, the model becomes more robust.

Representing Graphs as Tokens Each sample is structured as a list of edges, a list of nodes, the source and target nodes and then a list of nodes corresponding to the shortest path. We use a set of 6 control tokens to denote an edge, the beginning of the node list, the beginning of the query and the beginning of the shortest path in addition to standard beginning and end of sequence tokens. Each of the 10 node labels is a separate token thus, the vocabulary contains 16 total tokens. Please refer to Table 2 in Appendix A.2 for a complete description of the entire vocabulary.

Control tokens serve to explicitly identify the different components of a sample to relieve the model from needing to infer this from context. Additionally, control tokens serve as registers for the model to store intermediate results (Goyal et al., 2024; Darcet et al., 2024; Brinkmann et al., 2024).

Model We train two-layer decoder-only transformer models (Vaswani et al., 2017) with RoPE positional embeddings (Su et al., 2021). We use a fixed hidden dimension of 512 and train models with 1, 2, 4, and 8 heads. For a complete list of hyperparameters, please see Table 1 in Appendix A.1.

During training and testing, for each sample, we randomly shuffle the edge order, the node ordering within edges and the node ordering in the node list as well as relabel each node. This maintains the same graph structure but ensures the computations and representations the model learns will be robust to any specific labeling and ordering of a particular graph. In what follows, we refer to this procedure as a `remap`. Finally, we mask the loss on all but the tokens in the shortest path because the edge list, node list and source and target tokens are inherently unpredictable.

4 TRAINING RESULTS

In this section, we evaluate the model's ability to learn the task of generating shortest paths. As stated in Section 3, there may be multiple shortest paths between the source and target node. Thus, we present two results: the total probability assigned to all shortest paths and the distribution of path probabilities when many exist. For additional experimental results ablating model graph size, please see Appendix B.1.

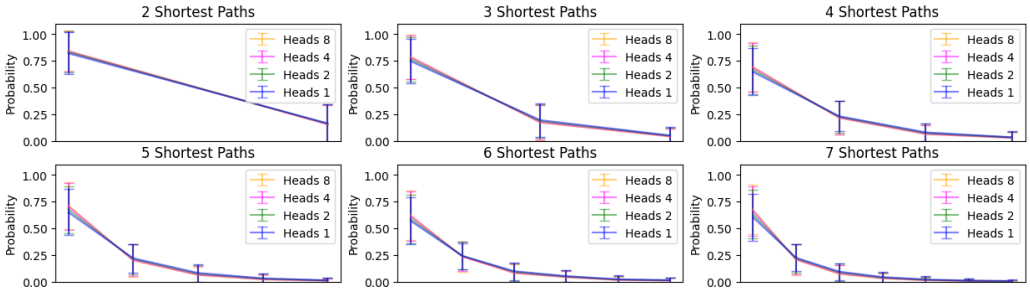

Figure 3: Probability distribution over $j \in \{2, 3, 4, 5, 6, 7\}$ shortest paths for 2 layer models with 1,2,4, and 8 heads. Samples are grouped by the number of shortest paths between source and target in the range. We compute the probability of each of the j paths and sort them in descending order. Each point is the mean and standard deviation over the test set.

4.1 ACCURACY

Figure 2 shows the probability of generating a shortest path resolved by the path length for 2-layer models with 1,2,4, and 8 heads on the test set. For a given graph and query, we compute all shortest paths (Hagberg et al., 2008) and compute the probability of sampling each path from the model with a temperature of $\tau = 0.7$. The sum of these probabilities is the probability that the model correctly generates a shortest path for a given graph and query. The distribution of probabilities over paths is presented in the next section.

In the top row of Figure 2, we observe that the number of heads is positively correlated with accuracy (although all model variants perform well and the worst category is the 1-head model on paths of length 6 where the 1.5 interquartile range is above 0.95). Additionally, we observe that path length is not necessarily correlated with difficulty.
In the bottom row of Figure 2, we characterize the samples which have a less than 80% chance of being solved by each model. In the failure cases, we observe that there usually exist many near-optimal paths (e.g., if the shortest path has length 4, there may be other paths of length 5) to which the model assigns non-trivial probability. For each sample, we take the 10 paths to which the model assigns the highest probability. Then, we separate the paths with length $\ell > \ell^*$ and compute the average $\bar{\ell}$.

We propose the following metric to measure the variation in path length:

$$\bar{\ell} - \ell^* \tag{1}$$

Note, $\bar{\ell} - \ell^* \geq 1$ because $\bar{\ell}$ is the average path length of paths $\ell > \ell^*$. We filter samples for $4 \leq \ell^* \leq 8$ as these are the samples with decreased performance. For each model, we bucket samples by the the probability assigned to the shortest paths in increments of 0.2 up to 0.8 and the value of $\bar{\ell} - \ell^*$. We plot the occurrences as color in the bottom row of Figure 2. These plots show there is a greater frequency of lower probability samples as the value of $\bar{\ell} - \ell^*$ approaches 1. Additionally, models become more robust to this effect with increasing heads.

4.2 DISTRIBUTION OF PATHS

In this section, we show that all model variants learn a *distribution* over paths even though we don't explicitly train the model on different paths for a given graph and query (up to a `remap`) over epochs.

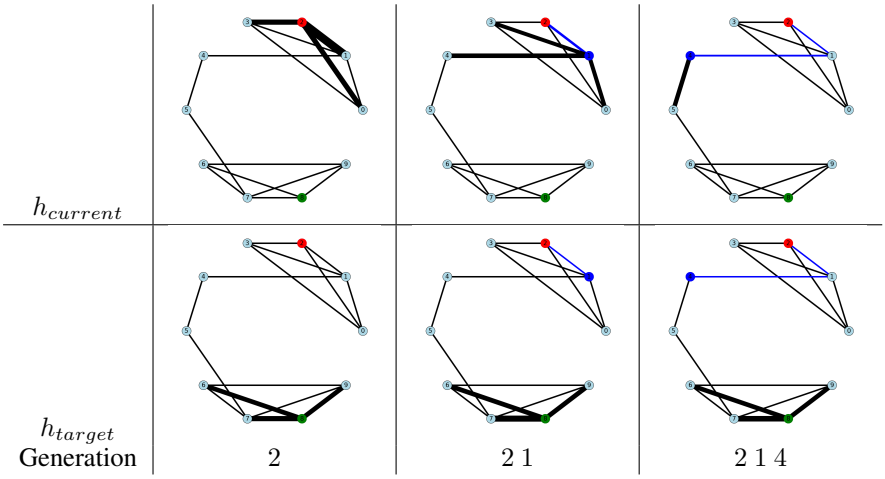

Figure 4: Attention activations of $h_{current}$ and h_{target} of the 4 head model in the final layer visualized as thickness of an edge for an example graph and shortest path query from source node (2) to target node (8). Each column corresponds to 1 iteration of generation and the current node and path so far is highlighted in blue. **(Top)** $h_{current}$ attends to the edge tokens corresponding to edges containing the current node in the sequence. Additionally, the relative attention activation is reduced for the edge connecting the current node to the previous node. **(Bottom)** h_{target} attends to the edge tokens corresponding to edges containing the target node.

All model variants exhibit a similar pattern - for each k, there is one path to which the model assigns $> 50\%$ probability and then a decaying but non-trivial probability to the rest of the paths. Although this behavior emerges without explicitly training for it, it is not clear that resampling the path over training epochs would drastically change the distribution and we leave this for future work.

We group samples by the number of paths $j \in \{2, 3, 4, 5, 6, 7\}$ with length ℓ^* and compute the probability for each path with each model. Then, we rank the path probabilities by descending order and average the probabilities over the rankings. We report these results in Figure 3.

5 Mechanistic Investigation

In this section, we provide two mechanistic results which we use to propose and implement *SLN* later on. Specifically, we show that the model learns attention heads which strongly attend to the edge control tokens $\langle e \rangle$ of edges which contain the current and target nodes in the second layer. Then, we examine the embeddings of the edge control tokens after the first layer and show that the model learns an embedding scheme wherein the distance between edge representations is correlated with the distances obtained by a spectral decomposition of $L(G)$. To obtain intermediate representations from the model, we use the Python package TransformerLens (Nanda & Bloom, 2022).

5.1 Attention Maps

In Section 4, we showed that increasing the number of heads correlated with improved accuracy. In this section, we examine attention activations as the model autoregressively generates a path between source and target node. We refer to the most recent intermediate node in the generated path as the *current* node.

We identify two distinct attention head mechanics in the second (and final) layer and denote them by $h_{current}$ and h_{target} and define their functions as:

- $h_{current}$: Attends to the edge control token $\langle e \rangle$ of edges which contain the *current* node in the shortest path.
- h_{target}: Attends to the edge control token $\langle e \rangle$ of edges which contain the *target* node in the shortest path.

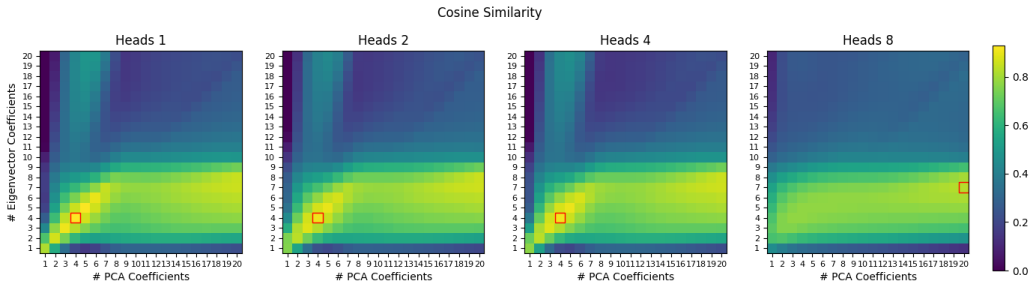

Figure 5: Cosine similarity of distance matrices between the top 20 principal components of edge token embeddings and the eigenvector coefficients of the 20 smallest non-zero eigenvalues of the normalized Laplacian of $L(G)$ over 10000 random samples. For each sample, we apply `remap` 100 times. For 1, 2 and 4 heads, the maximums (red square) are 0.928, 0.907 and 0.909, respectively with 4 eigenvector and 4 PCA coefficients. For 8 heads, the maximum is 0.826 with 7 eigenvector and 20 PCA coefficients.

To find $h_{current}$ and h_{target}, we manually inspect activation maps when the model is generating paths of at least 4 nodes. We focus on these paths, because the attention activations for paths with < 4 nodes are less peaked for $h_{current}$ and h_{target}, suggesting the model may use a potentially simpler algorithm in these cases (e.g., a 'lookup' table when the source and target nodes are connected by an edge). We select the heads from each model whose maximum activations correspond to $\langle e \rangle$ containing the current or target nodes. Note that the 2-head model does not need any selection, whereas the 8-head model learns redundant $h_{current}$ and h_{target}. Since the model distributes activation according to the degree of a node (which varies across samples), we normalize attention activations by dividing by the maximum value in a given sample. Additionally, other heads in the 4 and 8 head models attend to other control tokens as well as other edges in the graph, but we leave interpreting these activations to future work.

In Figure 4, we show the activations of $h_{current}$ and h_{target} (visualized as the thickness of an edge) for an example graph as the 4-head model generates the first 3 nodes in the shortest path. In the top row, $h_{current}$ attends to the edge tokens corresponding to edges containing the current node in the sequence. Additionally, the relative attention activation is reduced for the edge connecting the current node to the previous node, which we interpret as a mechanism to avoid returning to the previous node as this would not be the shortest path. In the bottom row, h_{target} attends to the edge tokens corresponding to edges containing the target node. We report the average normalized activations of $h_{current}$ and h_{target} on $\langle e \rangle$ versus other edges, respectively *at the first (source) node* for paths of length 4 or greater over the test set in Table 3 in Appendix B.2.

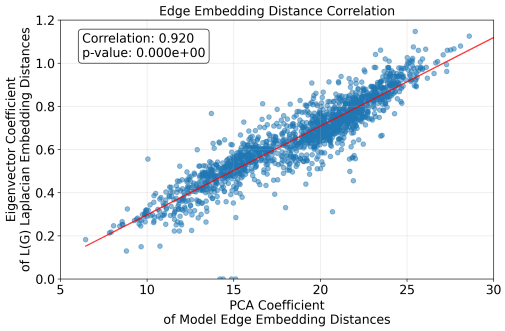

Figure 6: (**x-axis**) Unnormalized pairwise distance matrix of top 4 PCA coefficients of the embeddings of the control tokens $\langle e \rangle$ after the first layer of the 4-head model versus (**y-axis**) unnormalized pairwise distances between the eigenvector coefficients of edges corresponding to the smallest 4 non-zero eigenvalues of the normalized Laplacian of $L(G)$. Values are computed over all pairs of edges from 100 random samples and then we subsample 10% of values. The Pearson correlation coefficient is 0.92

5.2 Spectral Partitioning of $L(G)$ and Model Embeddings

Given the results in the previous section, we investigate the embeddings of the edge control token $\langle e \rangle$ after the first layer (as these are the representations to which the attention heads in the second layer attend). We assume that these embeddings represent specific edges (namely, the edge that precedes the control token). To interpret the algorithm learned by the model we ask: is there an algorithmic way to embed edges that comes directly from the graph theory. The most obvious guess is to consider spectral embeddings of the line graph Laplacian. Below we show that representations found by the model correlate strongly with the line

graph Laplacian. Specifically, we show that the principal components of the edge embeddings are correlated to the edge representations in the spectral decomposition of $L(G)$.

For a given sample, we apply `remap` 100 times and collect the activations of the residual stream after the first layer of the representation of $\langle e \rangle$, retaining the original edge orderings to do pairwise comparisons. For each `remap`, we compute the principal components separately to obtain a low-dimensional representation for each edge. We then average the pairwise distances between these representations over each `remap`. Alternatively, for the original graph, we compute the spectral decomposition of the normalized Laplacian of $L(G)$ and use the eigenvector coefficients to obtain a representation for each edge as is commonly done in spectral graph theory. These representations are completely independent of the model. This yields two vectors of length $\binom{n}{2}$ where n is the number of edges in the graph. We normalize both of these vectors to the range $[-1, 1]$ and compute their cosine similarity. Note cosine similarity is in the range $[-1, 1]$ where two vectors are more correlated as the value approaches 1.

We perform the above procedure for 10000 random samples from the test set and compute the cosine similarity when varying the number of eigenvector coefficients and PCA coefficients. We report the results in Figure 5. For 1, 2 and 4 heads, the maximums are 0.928, 0.907 and 0.909, respectively with 4 eigenvector and 4 PCA coefficients. For 8 heads, the maximum is 0.826 with 7 eigenvector and 20 PCA coefficients.

Figure 6 shows the Pearson correlation between the (*unnormalized*) pairwise embedding distances of the top 4 principal components of the 4-head model's edge token representations and the eigenvector coefficients of edges corresponding to the smallest 4 non-zero eigenvalues. These values are selected as they correspond to the maximum cell in Figure 5. We compute the pairwise distances over 100 randomly sampled graphs and then subsample 10% of points to plot. The Pearson correlation coefficient is 0.92. In simple terms, Figure 6 means that (on average) if two edges are close in model-embedding space then they are also close spectra-embedding space.

We note that the $\langle e \rangle$ embeddings are fundamentally incapable of perfectly matching the $L(G)$ decomposition due to the causal mask - the specific order of the edges may make it impossible for the model to represent the precise relative distances as they may become 'closer' due to a future edge to which the model cannot attend. It is possible that an encoder-decoder architecture (Vaswani et al., 2017) would be more successful but this is beyond the scope of this work as we want to understand the dynamics of decoder-only models.

6 Proposed Path-Finding Algorithm

In this Section we argue that the algorithm used by the model can be interpreted completely. Since we know that model essentially learns spectral embeddings, it is reasonable to ask: can we design a simple algorithm that leverages these embeddings (and possibly distances in the embedding space) to find the shortest path. We use the insights from Section 5 to propose a (novel) path-finding algorithm which we call Spectral Line Navigation (*SLN*). There exist shortest path algorithms which use the graph Laplacian (Steinerberger, 2020). To the best of our knowledge, *SLN* is the first which uses the Laplacian of $L(G)$.

There are three key components which follow from the results in Sections 5.1 and 5.2, respectively. Given a graph $G = (V, E)$, and source and target nodes, we first compute edge embeddings of the graph edges using the spectral decomposition of the Laplacian of $L(G)$. Then, beginning at the source node, we iteratively apply the following three steps until we reach the target node:

1. Gather the embeddings of edges containing the current node (i.e., $h_{current}$) and target node (i.e., h_{target}), constructing the sets $E_{current} = \{e_{current,i} \mid e_{current,i} \in E\}$ and $E_{target} = \{e_{target,j} \mid e_{target,j} \in E\}$.
2. Compute the L2 distance matrix $D^{current,target}$ between elements in $E_{current}$ and E_{target}

$$D^{current,target}_{i,j} = \|e_{current,i} - e_{target,j}\|_2$$
$$e_{current,i} \in E_{current}$$
$$e_{target,j} \in E_{target}$$

The i, j'th entry $D_{i,j}^{current,target}$ is the estimate of the distance in the graph between edges $e_{current,i}$ and $e_{target,j}$.

3. Select the edge $e_{current,i}$ with minimum distance to any of the the edges in E_{target}.

$$\hat{i} \leftarrow argmin_{i,j} D^{current,target}$$

Then, node $\hat{i}$ is the next node in the path.

We implement and run *SLN* on the test set and are able to achieve a final accuracy of 99.32%. We found that for most graphs (roughly 80%) using only the second smallest eigenvalue (i.e., the Fiedler vector from Section 2) for edge embeddings is sufficient. However, for other graphs we needed to increase the number k of non-zero eigenvalues we consider for edge embeddings. We report the log of counts for each $1 \leq k \leq |E|$ in Figure 9 in Appendix B.3. Additionally, please see Appendix B.1 wherein we predict and demonstrate with ablations that the model will fail to learn *SLN* as a function of the hidden dimension and maximum number of edges in the graphs. Additionally, please see Appendix B.1 wherein we predict and demonstrate with ablations that the model will fail to learn *SLN* as a function of the hidden dimension and maximum number of edges in the graphs.

7 Related Work

Interpretability and Reasoning Ideas from Mechanistic Interpretability have been applied to models trained on algorithmic data to understand the types of algorithms models may discover. Solutions learned by transformers trained on modular arithmetic have been fully reverse-engineered (Nanda et al., 2023) and the dynamics of feedforward networks have been theoretically characterized (Gromov, 2023; Tian, 2024; He et al., 2024). In transformers trained on data generated by iterative algorithms, the model learns a corresponding attention head which applies this iterative scheme (Cabannes et al., 2024). Circuits in transformers exist which can compose atomic subject-object relations into multi-hop relations across entities (Wang et al., 2024a) as well as compose variable relationships and assignments in propositional logic (Hong et al., 2024).

Graph Problems with Language Models Graph problems have served as a useful testbed for understanding the capabilities of language models with benchmarks such as GraphQA (Fatemi et al., 2024), CLRS-text (Markeeva et al., 2024) and NLGraph (Wang et al., 2023). Graph problems also have been used to illuminate numerous token representation limitations for solving combinatorial problems with language models (Ying et al., 2021; Perozzi et al., 2024; Markeeva et al., 2024; Bachmann & Nagarajan, 2024).

Path-finding with Language Models Interpreting language models trained on path finding tasks is of great interest as it is a problem that fundamentally requires planning and reasoning (Wang et al., 2024b). Models trained on directed acyclic graphs (Khona et al., 2024) to find any path, not necessarily the shortest, learn an algorithm that is very similar to *SLN* however it relies on node embedding distances instead of edge embedding distances, possibly due to differences in graph representation. There is no connection to the Graph or Line Graph Laplacian and we do not have any constraints on cycles in our setting. 6-layer decoder-only models trained to predict the path between source and target nodes in binary trees learn an iterative mechanism applied per layer whereas our model learns a embedding distance-based mechanism (Brinkmann et al., 2024). The SearchFormer line of work shows that encoder-decoder transformers can be trained to generate A^* search traces in order to navigate mazes and often generate smaller search trees than A^* itself (Lehnert et al., 2024; Su et al., 2024). Finally, there exist many alternate lines of work such as Graph Neural Networks (Wu et al., 2021) and Looped Transformers (De Luca & Fountoulakis, 2024).

8 Conclusion

In this work, we have investigated the problem of training to and understanding how decoder-only language models predict shortest paths. To that end, we have shown 2-layer models can learn this task with high accuracy and, by observing attention head dynamics and demonstrating a high correlation between token embeddings and the eigendecomposition of the normalized Laplacian of the line graph, proposed a novel path-finding algorithm Spectral Line Navigation. There are many possible directions for future work such as graph tasks beyond the shortest path, exploring the trade-off between model and graph size and alternative forms generalization besides unseen graph structures.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Anthropic. The claude 3 model family: Opus, sonnet, haiku. 2024.

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

Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, pp. 87–90, 1960.

Jannik Brinkmann, Abhay Sheshadri, Victor Levoso, Paul Swoboda, and Christian Bartelt. A mechanistic analysis of a transformer trained on a symbolic mulit-step reasoning task. *Association for Computational Linguistics*, 2024.

Viven Cabannes, Charles Arnbal, Wassim Bouaziz, Alice Yang, Francois Charton, and Julia Kempe. Iteration head: A mechanistic study of chain-of-thought. *Advances in Neural Information Processing Systems*, 2024.

Boris Cherkassky, Andrew Goldberg, and Tomasz Radzik. Shortest paths algorithms: theory and experimental evaluation. *Mathematical Programming*, 1996.

Fan Chung. Spectral graph theory. *American Mathemetical Society*, 1997.

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

Timothee Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. *International Conference on Learning Representations*, 2024.

Artur Back De Luca and Kimon Fountoulakis. Simulation of graph algorithms with looped transformers. *arXiv preprint arXiv:2402.01107*, 2024.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, pp. 269–271, 1959.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Benjamin Edelman. *Combinatorial Tasks as Model Systems of Deep Learning*. PhD thesis, Harvard University, 2024.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models, 2024.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Nagarajan Vaishnavh. Think before you speak: Training language models with pause tokens. *International Conference on Learning Representations*, 2024.

Andrey Gromov. Grokking modular arithmetic. *arXiv preprint arXiv:2301.02679*, 2023.

Aric Hagberg, Daniel Schult, and Pieter Swart. Exploring network structure, dynamics, and function using networkx. *Python in Science Conference*, pp. 11–15, 2008.

Frank Harary and Robert Norman. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 1960.

Tianyu He, Darshil Doshi, Aritra Das, and Andrey Gromov. Learning to grok: Emergence of in-context learning and skill composition in modular arithmetic tasks. *arXiv preprint arXiv:2406.02550*, 2024.

Guan Zhe Hong, Nishanth Dikkala, Enming Luo, Cyrus Rashtchian, Xin Wang, and Rina Panigrahy. How transformers solve propositional logic problems: A mechanistic analysis. *arXiv preprint arXiv:2411.04105*, 2024.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

Mikail Khona, Maya Okawa, Jan Hula, Rahul Ramesh, Kento Nishi, Robert Dick, Ekdeep Singh Lubana, and Hidenori Tanaka. Towards an understanding of stepwise inference in transformers: A synthetic graph navigation model. *arXiv preprint arXiv:2402.07757*, 2024.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping, 2024.

Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, 2022a.

Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *The Eleventh International Conference on Learning Representations*, 2022b.

Larisa Markeeva, Sean McLeish, Borja Ibarz, Wilfried Bounsi, Olga Kozlova, Alex Vitvitskyi, Charles Blundell, Tom Goldstein, Avi Schwarzschild, and Petar Veličković. The clrs-text algorithmic reasoning language benchmark. *arXiv preprint arXiv:2406.04229*, 2024.

B. D. Mckay. Simple graphs. URL `https://users.cecs.anu.edu.au/~bdm/data/graphs.html`.

B. D. Mckay. Applications of a technique for labelled enumeration. *Congressus Numerantium*, pp. 207–221, 1983.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathemetical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

Neel Nanda and Joseph Bloom. Transformerlens. `https://github.com/TransformerLensOrg/TransformerLens`, 2022.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress meausres for grokking via mechanistic interpretability. *Internation Conference on Learning Representations*, 2023.

Eshaan Nichani, Alex Damian, and Jason D Lee. How transformers learn causal structure with gradient descent. *arXiv preprint arXiv:2402.14735*, 2024.

Maya Okawa, Ekdeep S Lubana, Robert Dick, and Hidenori Tanaka. Compositional abilities emerge multiplicatively: Exploring diffusion models on a synthetic task. *Advances in Neural Information Processing Systems*, 36, 2024.

Yonatan Oren, Nicole Meister, Niladri Chatterji, Faisal Ladhak, and Tatsunoria Hashimoto. Proving test set contamination in black box language models. *arXiv preprint arXiv:2310.17623*, 2023.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for llms, 2024. URL `https://arxiv.org/abs/2402.05862`.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Stefan Steinerberger. A spectral approach to the shortest path problem. *Linear Algebra and its Applications*, 2020.

Di Jia Su, Sainbayar Sukhbaatar, Michael Rabbat, Yuandong Tian, and Qinqing Zheng. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. *arXiv preprint arXiv:2410.01779*, 2024.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. arxiv e-prints, art. *arXiv preprint arXiv:2104.09864*, 2021.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Yuandong Tian. Composing global optimizers to reasoning tasks via algebraic objects in neural nets. *arXiv preprint arXiv:2410.01779*, 2024.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. *Advances in Neural Information Processing Systems*, 2024a.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=UDqHhbqYJV.

Siwei Wang, Yifei Shen, Shi Feng, Haoran Sun, Shang-Hua Teng, and Wei Chen. Alpine: Unveiling the planning capability of autoregressive learning in language models. *Conference on Neural Information Proessing Systems*, 2024b.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural newtorks. *IEE Transactions on Neural Networks and Learning Systems*, 32:4–24, 2021.

Sohee Yang, Elena Griboskaya, Nora Kassner, Mor Geva, and Sebastion Riedel. Do large language models latently perform multi-hop reasoning. *Association for Computational Linguistics*, 2024.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of Language Models: Part 2.1, Grade-School Math and the Hidden Reasoning Process. *ArXiv e-prints*, abs/2407.20311, July 2024. Full version available at http://arxiv.org/abs/2407.20311.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 28877–28888. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f1c1592588411002af340cbaedd6fc33-Paper.pdf.

A Training Details

A.1 Hyperameters

Field	Value
layers	2
num_heads	$\{1, 2, 4, 8\}$
hidden_dim	512
MLP hidden_dim	2048
vocab_size	16
RoPE θ	10000.0
weight_decay	0.001
optimizer	adamw
β_1	0.9
β_2	0.99
lr_scheduler	cosine annealing
lr_ratio	.1
warmup_epochs	1
epochs	2000

Table 1: Training Parameters

A.2 Control Tokens

Token	Description
$[0-9]$	Nodes
$\langle bos \rangle$	Beginning of sequence
$\langle eos \rangle$	End of sequence
$\langle e \rangle$	An edge between the previous two node tokens
$\langle n \rangle$	The beginning of the node list and end of the edge list
$\langle q \rangle$	Beginning of the query. The following two node tokens are the *source* and *target* nodes
$\langle p \rangle$	Beginning of the path. The following nodes are the nodes in the shortest path beginning with the *source* and ending with the *target*

Table 2: **Vocabulary**

B Additional Results

B.1 Ablations

In this section, we present two additional results. Given the algorithm *SLN* which we have identified, we are able to make a prediction about the model size required to learn the task of predicting shortest paths. Specifically, since the Laplacian of the line graph is an NxN matrix where N is the number of *edges* in the graph, to fully represent this matrix, the hidden dimension of the model must be greater than N. In our setting, graphs contain up to 10 nodes, so the minimum hidden dimension we would predict to learn the task is $\binom{10}{2} = 45$. We test this prediction in Figure 7 with the same experimental setting described in Section 3. 2-layer models with 4 attention heads and hidden dimension of 64 and greater are able to reduce the loss and learn the task. However, the 2-layer model with a hidden dimension of 32 fails. We also point out that the greater the hidden dimension, the more quickly it can reduce the loss.

A natural follow up experiment is to ablate the maximum number of edges in graphs in the training set. We present these results in Figure 8. The 2-layer model with hidden dimension of 32 is able

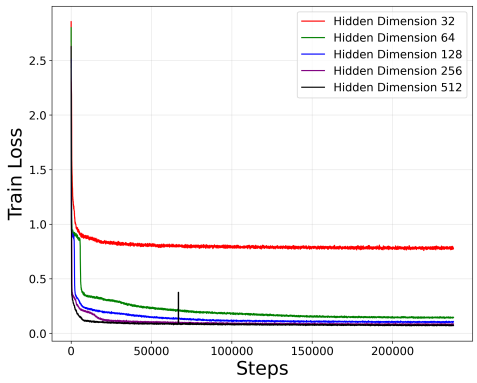

Figure 7: Ablation of the hidden dimension of the 2-layer model with 4 attention heads. We predict that the model will fail when the maximum number of edges in a graph (45) exceeds the hidden dimension. The model with hidden dimension of 64 and greater are able to reduce the loss and learn the task but 32 is not.

to reduce the loss if we bound the maximum number of edges (e.g., 15 and 25) but at 35 edges the model is still unable to learn.

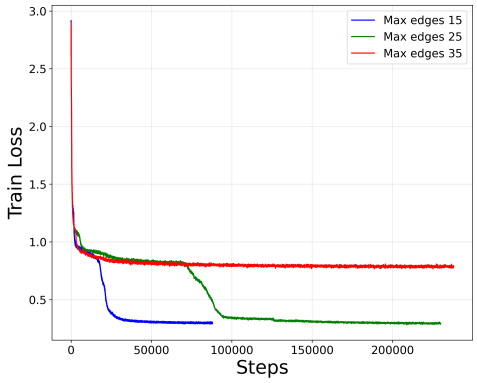

Figure 8: Ablation of the maximum number of edges in graphs in the training set. When we bound the number of edges to be less than the hidden dimension, the model is able to reduce the loss.

B.2 Attention Heads

We present additional results to complement the results in Section 5.1. We find $h_{current}$ and h_{target} by manually expecting attention maps of each head in each model. Since the model distributes activation according to the degree of a node (which varies across samples), we normalize attention activations by dividing by the maximum value in a given sample. In Table 3, we present the activations of $h_{current}$ and h_{target} averaged over the test set.

	Head 2 (/2)	Head 4 (/4)	Head 5 (/8)
Current	0.826	0.75	0.779
Other	0.284	0.029	0.026
Ratio	2.91	25.86	29.96
	Head 1 (/2)	Head 1 (/4)	Head 1 (/8)
Target	0.865	0.868	0.561
Other	0.45	0.036	0.007
Ratio	1.92	24.11	80.14

Table 3: Table showing that each model learns heads which attend to current node/target node edges

B.3 Spectral Line Navigation

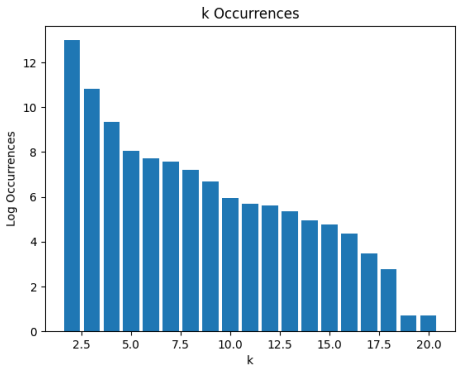

Figure 9: The log of k counts needed for *SLN* to successfully find the shortest path over the test set. For most graphs, using only the second smallest $k = 1$ (i.e., Fiedler Vector) is sufficient.

We implement and run *SLN* on the test set and are able to achieve a final accuracy of 99.32%. We found that for most graphs (roughly 80%) using only the second smallest eigenvalue (i.e., the Fiedler vector from Section 2) for edge embeddings is sufficient. However, for other graphs we needed to increase the number k of non-zero eigenvalues we consider for edge embeddings. We report the log of counts for each $1 \leq k \leq |E|$ in Figure 9.