Discovering Mechanisms in Tokenized Graph Transformers

Yong-Min Shin [1] **Dae-Woong Jeong** [1] **Sehui Han** [1]

Abstract

We investigate the internal mechanisms of a tokenized graph transformer — a T5 encoder trained on graphs represented as sequences of node and edge tokens — to understand how transformers process graphs and solve tasks. Using mechanistic interpretability tools such as activation patching and linear probing, we aim to understand the model under three fundamental graph tasks: degree counting, ring membership, and shortest-path distance. Our analysis reveals a common early local-structure computation, where degree-like features emerge in shallow layers and are directly used to solve degree counting. Then, the model composes this early signal differently according to each task. In ring membership, we find that the model solves this problem by gathering non-ring node evidence rather than building a cycle detection circuit. In the shortest-path distance task, causal evidence supports a serial pipeline in which early local topology feeds a single adjacency-copy head, followed by later refinement of a distance-like representation. Furthermore, we analyze the QK circuit underlying this behavior and show that early layers implement soft node-incidence tests by matching node IDs to edge endpoint IDs. Finally, we discuss our limitations and potential future research directions.

1. Introduction

Mechanistic interpretability (Bereska & Gavves, 2024) has made substantial progress by reverse-engineering internal computations in transformer models by developing various tools of investigation (*e.g.*, activation patching (Meng et al., 2022; Zhang & Nanda, 2024)). Understanding how the model interacts with a given input not only reveals how the model decodes complex information, but also opens up potential applications where the user can directly control the model at inference (Lieberum et al., 2024). However, much of this progress has focused on language, while mechanistic understanding in other domains remains underdeveloped.

As an expansion of mechanistic analysis to broader domains, we turn our attention to *graph inputs*, *i.e.*, nodes and edges. Graphs naturally encode rich and complex information (*e.g.*, molecular structures), and learning the ability to utilize structural information is the key to solving downstream tasks. To achieve this, a graph learning model must be able to perform permutation-symmetric computations (equivariant or invariant, depending on the task) (Bronstein et al., 2021) *and* solve the task. Unlike pixels in images and words in natural language where canonical ordering naturally exists, nodes in a graph are inherently unordered. In other words, a single graph can be represented multiple ways by shuffling its node identifiers (IDs). As a result, the ability to understand the input graph regardless of node permutations is an essential and unique requirement for graph-based models.

Unlike graph neural networks (Gilmer et al., 2017; Wu et al., 2021) that handle this requirement intrinsically, graph transformers (Müller et al., 2024) do not have this feature baked into the architecture. Although many graph transformers alleviate this by adding various graph biases, it is known that *vanilla transformers*–widely investigated in many mechanistic interpretation works–can learn this behavior and solve graph learning tasks as long as nodes and edges are given as explicit tokens (Kim et al., 2022).

In this work, we focus on the following question: **"How do vanilla transformers perform graph computation given explicit node and edge tokens?"**. To this end, we analyze transformers trained to solve fundamental graph tasks (degree counting, ring membership, and shortest-path distance) while constantly permuting node IDs during training to make the model permutation equivariant. We find (and provide multiple evidence) the following three themes when aggregating our understanding across tasks:

1. **Early degree-like computation:** Across tasks, the earliest robust structural signal is a distributed degree-like computation in layer 0.
2. **Task-specific composition:** Harder tasks build on early computation according to different tasks. De-

[1]LG AI Research, Seoul, South Korea. Correspondence to: Yong-Min Shin <yongmin.shin@lgresearch.ai>.

Mechanistic Interpretability Workshop at the 43rd International Conference on Machine Learning, Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

gree counting reads off of a degree direction in the residual stream. Ring membership uses non-ring evidence, while shortest path uses a BFS-parent copy head followed by distance refinement.

3. **Node ID identification:** Before solving the task, the model must identify which edge tokens are attached to each node token. We find that L0 attention heads do this by giving higher scores to edge tokens whose endpoint ID matches the node ID. This behavior appears across all three task models, showing that the model first learns a general incident-edge retrieval step at layer 0, which is crucial for permutation-aware computations.

Although the field of mechanistic interpretability research is actively expanding, understanding the internals of *graph* transformers is still mostly overlooked. Notably, (Sanford et al., 2024) investigated the transformer's ability to solve graph tasks, but they are more focused on architectural requirements rather than intrinsic computation. A recent work (El et al., 2025) has focused on the internals of graph transformers, but it focused on attention distributions and did not investigate vanilla transformers. To the best of our knowledge, our work is the first attempt to understand the mechanistic structure of a vanilla transformer on graphs.

The rest of the paper's organization is as follows. We first describe the model and task setup in more detail. We then introduce the common early degree-like computation, analyze how the model solves each task, and finally study the QK circuits that recover graph incidence from tokenized node and edge IDs. Finally, we conclude with discussions on the limitations of our work, alongside the appendix for further details.

2. Model and Task Setup

2.1. Dataset and Task

For the dataset, we created a synthetic dataset composed of undirected graphs with $N \in [16, 32]$ nodes. All tasks share the same base corpus with splits (25,000 train / 5,000 val / 5,000 test), stratified by node count and cycle rank.[1] A Weisfeiler-Lehman graph hash (Shervashidze et al., 2011) was used to ensure no structurally isomorphic graphs were leaked. The validation set was used to check training convergence, and the test set for mechanistic analysis.

2.2. Graph Representation and Tokenization

When vanilla transformers are concerned, how we represent the graph determines how the model interprets the graph and therefore becomes a critical design choice. Following TokenGT (Kim et al., 2022), graphs are represented by explicit tokenizations of the nodes and edges via type indicators and node IDs in our work. Specifically, a graph is represented as a sequence of tokens by the following procedure. First, we pre-construct fixed type vectors $\tau_{\text{node}}, \tau_{\text{edge}} \in \mathbb{R}^{16}$ and node ID vectors $\{e_i \in \mathbb{R}^{64} | i = 0, \cdots, 31\}$. All vectors within each set are randomly sampled from a uniform distribution and further processed to be mutually orthonormal. Then, a node v is represented as $[\tau_{\text{node}} \| e_v \| e_v] \in \mathbb{R}^{144}$.[2] Similarly, an edge (u, v) connecting node u and v is represented as $[\tau_{\text{edge}} \| e_u \| e_v] \in \mathbb{R}^{144}$. By constructing such token sequence, all information necessary to fully represent the graph structure is fed into the model, which then gets projected via $W_{\text{proj}} \in \mathbb{R}^{256 \times 144}$ into the residual stream.

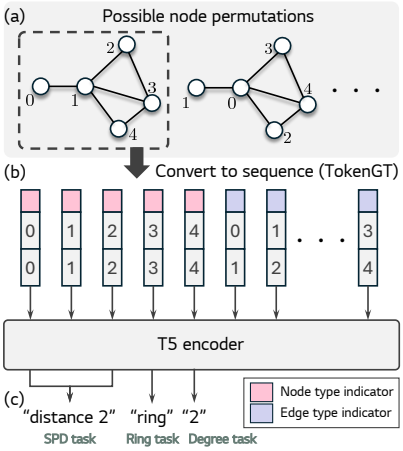

Figure 1. Model and task setup. (a) The same graph can be represented multiple ways by permuting node IDs. (b) We consider the case where nodes and edges are tokenized by assembling type and ID vectors (Section 2.2). (c) Different heads are applied to the node outputs to solve graph tasks. Note that we train *separate* models per task and the figure is a conceptual description.

2.3. Task Specification

Here, we describe the three tasks used in our experiments, where the model is expected to produce predictions per node or node pairs.

Degree counting (DC). Node classification task where the correct label is the number of neighboring nodes connected by an edge. Here, class 0∼3 represents degree 1∼4, and class 4 means degree ≥ 5.

Ring membership (RM). Binary node classification task where the correct label is whether the node in question belongs to any ring structure.

Shortest-path distance (SPD). Node-pair classification task where the label is assigned as the shortest path distance between node u and v. Here, class 0∼3 represents

[1] For SPD, we use the largest component subset (∼80%).

[2] We denote $\cdot \| \cdot$ as concatenation.

distance 1∼4, class 4 means distance ≥ 5.[3]

Although all three tasks are commonly used throughout the graph learning literature (Yan et al., 2023; Ying et al., 2021), graph-level tasks such as graph counting are not included, which we leave as future work.

2.4. Model and Training

We used the encoder-only T5 model (Raffel et al., 2020) with task-specific heads in this study, closely following TokenGT (Kim et al., 2022). Across all tasks, the model has $L = 4$ layers, model dimension $d_{\text{model}} = 256$, $H = 8$ attention heads per layer, $d_{\text{kv}} = 32$, and MLP dimension $d_{\text{ff}} = 2048$ without positional encoding.[4]

For all three tasks, we train the model with AdamW (Loshchilov & Hutter, 2019) with learning rate 1e-5, weight decay 1e-4, and batch size 2048. Importantly, we train the model **with node ID permutation**. In other words, we randomly shuffle the node IDs before the model for every iteration during training to ensure the trained model learns permutation equivariance (*i.e.*, the model learns to understand graph structure under any node ID assignments). For all three tasks, **the model was trained successfully** (100%, 100%, and 99.4% validation accuracy for degree counting, ring membership, and shortest-path distance, respectively). For the rest of the section, we assume all analyses are performed on these models unless explicitly stated otherwise.

We summarize how we address specific parts in the model. We use 'Lℓ-pre', 'Lℓ-mid', and 'Lℓ-post' for the residual stream before attention, after attention/before the MLP, and after the MLP in layer ℓ, respectively. $z_i^{(\ell,h)}$ denotes the output of head h at token i before the output projection $W_O^{(\ell,h)}$. Finally, 'Lℓ:Hh' denotes head h in layer ℓ.[5]

2.5. Evaluation

Unless otherwise stated, all results from this section onward are the result of performing various analysis methods using 100 randomly sampled graphs from the test set, which the model has not seen previously.

3. Degree Computation at Early Layers

We first discuss a common pattern that emerges across all three tasks, which is that earlier layers focus on *extracting local degree-like features* as a specific direction in the residual stream. To back this claim, we first describe how the degree direction is found, and perform an experiment along the degree direction and observe the model's performance as a consequence of this intervention.

Table 1. Intervention results along the degree direction. For each task (and model), we identify the degree directions (L0 for DC and RM, L1 for SPD) via linear probing and directly apply activation patching on the residual stream by adding $\alpha\, \hat{w}_{\text{deg}}^{(l)}$ ($l = 0$ for DC and RM, $l = 1$ for SPD) directly and observing the performance.

α	Mean degree shift	Ring/Non-ring acc.	SPD acc.
-5	-0.9125	0.5513/1.0000	0.9612
-3	-0.5437	0.9708/1.0000	0.9898
0	$+0.0000$	1.0000/0.9986	0.9891
$+3$	-0.1958	1.0000/0.8622	0.9116
$+5$	$+0.2271$	1.0000/0.0000	0.7539

How the degree direction is defined. To find the degree direction, we fit a linear probe on the L0-mid activations $r_{\text{mid},i}^{(0)} \in \mathbb{R}^{d_{\text{model}}}$ from the 100 test graphs randomly sampled from the test set. Denoting $\bar{r}_{\text{mid},i}^{(0)}$ as the normalized activation on the test split, we can write the probe as $f(r_{\text{mid},i}^{(0)}) = \text{softmax}(B\bar{r}_{\text{mid},i}^{(0)} + b)$ with $B \in \mathbb{R}^{5\times d_{\text{model}}}$ and $b \in \mathbb{R}^5$ that solves the degree classification problem.[6]

We define the **degree direction** as

$$\hat{w}_{\text{deg}}^{(0)} = \frac{\tilde{B}_{4,:} - \tilde{B}_{0,:}}{\|\tilde{B}_{4,:} - \tilde{B}_{0,:}\|_2}, \tag{1}$$

where $\tilde{B}_{c,:}$ are the rows in B normalized by the standard deviation of $r_{\text{mid},i}^{(0)}$.[7] Note that $\hat{w}_{\text{deg}}^{(0)} \in \mathbb{R}^{d_{\text{model}}}$ is a residual-stream direction.

Intervention along the degree direction. Table 1 shows the key intervention results by adding $\alpha\, \hat{w}_{\text{deg}}^{(l)}$ in the residual paths with different α values. We see the most direct impact for the DC task, where steering in the positive and negative direction makes the model change the prediction accordingly. Specifically, adding $-5\, \hat{w}_{\text{deg}}^{(0)}$ at the L0-mid activation $r_{\text{mid}}^{(0)}$ shifts downstream predictions by -0.91 classes on average, strongly suggesting that the degree probe direction is read by the downstream weights. In the RM task, the degree direction is also causally active on the ring/non-ring nodes: intervening by $+5\, \hat{w}_{\text{deg}}^{(0)}$ collapses the non-ring accuracy to 0, but steering in the opposite direction does not harm the accuracy (and vice versa). In the SPD task, a separate de-

[3] Self-pairs are not considered and excluded from training loss.

[4] This is natural since there is no natural node ordering, and therefore token positions do not convey meaningful information.

[5] For further details on model architecture and exact probing/ablation points, refer to Appendix A.

[6] Specifically, $\bar{r}_{\text{mid},i}^{(0)} = (r_{\text{mid},i}^{(0)} - \mu) \oslash \sigma_x$, where μ and σ_x are the mean and elementwise standard deviation, and $\oslash$ denotes elementwise division.

[7] We denote the degree direction for layer 0 as a representative (i.e., $\hat{w}_{\text{deg}}^{(0)}$) as the degree direction exists from layer 0 for most cases. However, we use $\hat{w}_{\text{deg}}^{(1)}$ for the SPD task later.

gree direction extracted at $r^{(1)}_{\text{post}}$ is causally active: adding $+5\,\hat{w}^{(1)}_{\text{deg}}$ reduces pairwise accuracy from 0.989 to 0.753.

These results support two claims: (1) degree-like features are computed in earlier layers, and (2) the more complex tasks (*i.e.*, RM and SPD) use these features directly in the later computations. We discuss **how** these features are used by later layers of each model in the following sections.

4. Degree Counting: Magnitude As Features in the Degree Direction

4.1. Task is Already Solved at Layer 0, Later MLPs Refine Them

As seen from the degree direction results in Table 1, we can expect that the DC task is already solved at layer 0. Indeed, separately training a linear probe for different positions in the early residual stream shows that the information necessary to solve DC is already linearly decodable at layer 0. Specifically, the results in Table 2 show that degree information is not present at embed level, then becomes almost linearly recoverable after the first attention pass.

This also suggests that the later layers are either refinements or redundancies. Zero-ablating for each attention and MLP modules reveals that the early MLP layers are likely acting as refinements, while the later layers (*i.e.*, L2 and L3) only drop the accuracy around $\sim 1\%$ (refer to Appendix B for detailed results and Appendix F.1 for additional discussions).

To further validate the effect of MLPs on refinement, we directly observe the average class-wise distance before and after L0 MLP is applied in the residual stream. We observe the mean distance between the centroids has increased to 4.27 post MLP, an $\times 1.74$ amplification from the 2.46 before MLP. Essentially, this makes the degree features inside the model more stable since the classification boundaries are more cleanly separated. This effect is also directly observable by revisiting the degree direction intervention again, but comparing with/without L0 MLP. We can expect the model becomes more sensitive to the intervention when L0 MLP is zero-ablated as there are no refinements to counteract the intervention, which is what we observe in Figure 2.

4.2. Degree-Proportional Writing in L0

In this section, we look into more depth on how the first attention layer solves the DC task. We provide two directions of evidence: First, we show that most of the attention weights are focused on the incident edges themselves. Then, we also show how much each head writes in the degree direction, and show that writing in the degree direction is a collective effort, which makes the model resistant to head-wise ablations.

Table 2. Linear probe accuracy for node degree at successive residual-stream positions in the degree model. The DC task is nearly solved at the first attention pass.

Position	Degree Probe Accuracy
Embed	0.396
L0 mid	**0.973**
L0 post	0.994
L1 post	1.000

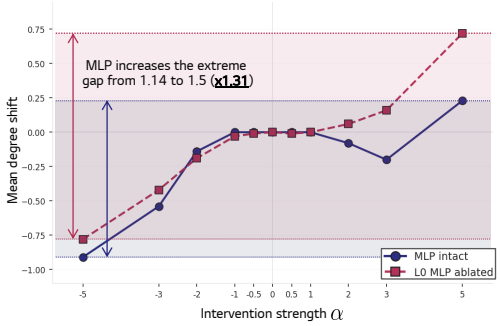

Figure 2. The model's sensitivity with and without L0 against degree direction intervention. The intervention sensitivity increases roughly $\times 1.31$ times when L0 MLP is zero-ablated.

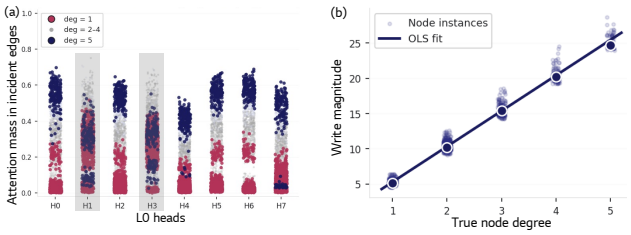

Figure 3. Detailed degree writing behavior in L0. (a) Attention mass allocated to the incident edges. Aside from H1 and H3 (gray blocks), the rest of the attention heads allocate their attention mass proportional to the number of incident edges needed to be attended (*i.e.*, degree). (b) The inner product between output projections of the attention block in L0 and the degree direction. Results show extremely high correlation (R^2=0.99), signifying the write magnitude represents the degree information.

Attention strongly favors neighboring edges. To observe whether the attention weights correctly aggregate from each node token's incident edge tokens, we directly measure the per-head attention mass focused on the incident edges by varying degree in Figure 3(a). The figure shows that 6 out of 8 heads allocate more aggregated attention mass for higher degree nodes. This suggests that the degree information is encoded as the magnitude in the degree direction, since high degree nodes have more incident edges to attend to (*i.e.*, a node of degree 5 has five edge tokens compared to node of degree 1 only having one edge token to attend). [8]

[8] We also compare the mean attention weight to incident-edge positions versus nonincident positions. Ratios of $5.85\times$–$17.82\times$ across all eight heads support a soft incident-edge selector via identifier-based QK content matching (refer to Appendix C for the detailed result).

Table 3. Ring/non-ring node behavior under L1 zero-ablation. The performance drop is severe for non-ring nodes but nearly unaffected for the ring nodes, even for larger rings.

Ring size	Logit margin	Accuracy	Δ accuracy
3	$+12.2$	100.0%	-0.0%
4	$+11.8$	99.0%	-1.0%
5	$+11.4$	99.1%	-0.9%
6	$+9.8$	94.4%	-5.6%
7	$+11.4$	95.5%	-4.5%
8	$+9.3$	100.0%	-0.0%
15	$+1.8$	100.0%	-0.0%
Non-ring	–	63.6%	$\mathbf{-36.4\%}$

How much does each head write? To directly observe each head's contribution to writing the degree direction in L0, we perform a per-head observation. Specifically, for each node i and each L0 head h, we extract the per-head pre-output vector $z_i^{(0,h)}$, and measure the class-wise inner product between the degree direction $\hat{w}_{\text{deg}}^{(0)}$ and the output projection of the pre-output vector $W_O^{(0,h)} z_i^{(0,h)}$. In other words, we measure $\mathbb{E}_{i:\deg(i)=d}\left[\left\langle W_O^{(0,h)} z_i^{(0,h)}, \hat{w}_{\text{deg}}^{(0)}\right\rangle\right]$. As shown in Figure 3(b), the write magnitude is very highly correlated with the true node degree with $R^2 = 0.99$.

Combined, we have a strong empirical set of evidence that the transformer model solves DC in the following manner: A node with identifier $[i, i]$ matches edge tokens $[u, v]$ where $u = i$ or $v = i$ (incident edge tokens). The degree signal emerges because (a) attention mass on incident edges scales with degree, and (b) the W_V-projected value of each attended edge token contributes constructively along the degree direction. The processed feature is read by the DC task head, while other later layers contribute minimally.

5. Ring Membership: Non-Ring Evidence Accumulation

Next, we attempt to identify how the model solves the RM task. On the surface, it seems likely that the task requires explicit cycle detection to be solved, and naturally one might expect the model to trace closed paths or maintain a multi-hop cycle circuit. But the evidence points to a different and more distribution-specific mechanism. Based on our experiments, the model appears to classify **all nodes as ring by default** in the earlier layers, then **accumulates evidence that a node is not on a ring**.

5.1. L1 Reads Necessary Information From L0 and Nearly Solves the Task

Firstly, we localize our analysis to L1 since training linear probes to solves the RM task in different layers show that task-critical information emerges at L1 attention, not at L0 (accuracy increases from 0.835 at L0-post to 0.974 at L1-mid). This goes against the initial expectation that the model will learn cycle-tracing circuits (which is likely to require deep layers). Therefore, we perform additional analysis at the L0-post position $r_{\text{post}}^{(0)}$, where the task-critical computation is likely to be performed, in order to identify *what* L1 reads.

5.2. The Model Assumes Ring by Default

Revisiting the degree direction intervention. The first hint of how the model actually solves the RM task can be found in the degree direction intervention experiment. Revisiting Table 1, we see that degree information is likely to be directly used by the model. However, the performance degradation is the opposite for ring and non-ring nodes. This can be partially explained if the following are true:

- Claim 1: The model assumes all nodes to be included in a ring by default.
- Claim 2: The model uses **leaf-neighbor evidence** (*i.e.*, degree 1 nodes) as primary sources of non-ring membership evidence since leaf nodes cannot be part of any ring and likely will be their neighbors under the data distribution.

To provide evidence that this **is** a valid approach to solve the RM task, we explicitly wrote an algorithm following this claim, which achieves up to 95.4% accuracy under our data distribution (see Appendix E). We perform an additional set of experiments to support these claims.

The model assumes ring-node by default. To support the first claim, we perform zero-ablation on L1 to understand how this affects the model prediction. Table 3 shows that the L1 zero-ablation mostly affects non-ring nodes, suggesting that most of the predictions before L1 are likely towards positive ring membership, but L1 overrides this prediction if some non-ring evidence is gathered. The table also shows that the model can solve for larger rings (ring sizes up to 15). If the model actually builds a cycle-tracing circuit, a deeper model should be required to identify rings of such large sizes. However, if the model assumes ring nodes by default, the results are explainable as larger rings are not affected since the model will fail to gather evidence that it is not a ring due to the model's shallow layer depth.

Non-ring signal is mostly leaf-neighbor evidence. For the second claim, we further look into the behavior of L1 by selecting the L1:H2 head as a representative, which has the strongest attention allocation towards incident edges (70.5% of attention mass on edge tokens, and 66.9% of those edge attention on incident edges). To understand what information is encoded, we train a series of linear probes in L1:H2's $z_i^{(1,2)}$.

Table 4. Various ablation results on L1 and its effect on ring/non-ring classification accuracies.

	Intervention	Ring	Non-ring
(a)	Clean model	1.000	0.999
(b)	Zero N→E	**0.693**	0.987
(c)	Zero N→N	1.000	0.955
(d)	Zero self	1.000	0.959
(e)	Uniform N→E on incident edges	1.000	0.999
(f)	Same as (e), but zero N→N + self	1.000	**0.300**

From the experiments, we find that the strongest decoded content was **whether one of its neighbors is a leaf** (*i.e.*, degree 1 neighbor). Specifically, we trained a linear probe to solve the following for a node token i: Given $z_i^{(1,2)}$, predict whether i has at least one leaf neighbor, *i.e.* whether there exists $j \in \mathcal{N}(i)$ with $\deg(j) = 1$, resulting in 0.867 accuracy.[9] Furthermore, applying ordinary least squares (OLS) resulting in a R^2=0.681, suggesting that the leaf-neighbor evidence is linearly encoded in $z_i^{(1,2)}$.

5.3. Channel Isolation: Ring and Non-Ring Evidence Are Separable

We further identify that the ring confirmation and non-ring evidence identification are performed separately in L1. By design, each layer in the model has three relevant node-query channels: node-to-edge (N→E), node-to-node (N→N), and self-attention. We ablate the attention in L1 for all heads for different node-query channels (and renormalizing the remaining attention weights to 1) to observe the performance for ring/non-ring nodes.

From the results in Table 4, we notice that experiment (b) and (f) indicate L1 has two complementary parts. The N→E channel is primarily responsible for the ring-confirmation signal from incident edges. The N→N + self channels are responsible for providing the non-ring evidence and overriding the default ring prediction.

5.4. Summary

In summary, the model uses the same degree-like feature computation combined with a broad node-to-node/node-to-edge channel to convey leaf-node information for non-ring evidence. However, we could not find any fine-grained internal processing mechanisms for L1 or other layers.

6. Shortest Path Distance: Strong Patterns of Adjacency Copy-Routing

We now identify how the SPD model uses local graph information to predict a pairwise distance. We find that the model does not appear to implement an exact shortest-path algorithm. Instead, it builds local graph strucutre summaries in early layers, uses L2:H2 as a BFS-parent copy head, and then lets L3 and the bilinear task head refine the resulting pair representation. This section first asks what information is available before L2:H2, then asks why removing or rerouting the BFS-parent copy hurts performance, and finally analyzes how L3 and the task head read out the longer-distance classes.

Table 5. Probe accuracies for local and rooted diagnostic features at the embedding and L1-post positions.

Feature	Embed	L1-post
Degree	0.405	0.942
Leaf neigh. count	0.737	0.824
High-degree neigh. count	0.566	0.842
BFS depth	0.408	0.492
BFS child count	0.518	0.687

6.1. What Local Features Are Constructed in Early Layers?

Degree is already known to be constructed early in the SPD model (see Table 1), but degree alone cannot solve a pairwise shortest-path task. We therefore probe several local and rooted graph variables at the embedding and L1-post positions. The local variables are computed directly from the graph neighborhood. The rooted variables are computed from a BFS tree rooted at the graph's highest closeness centrality node (*i.e.*, the node with the lowest average distance to all other nodes).[10]

Table 5 shows that L1-post already contains several local graph structure summaries. Degree, leaf-neighbor count, and high-degree-neighbor count are all linearly decodable by L1-post. By contrast, the rooted variables (BFS depth and BFS child count) are weaker at L1-post. Thus, before L2:H2, the model has strong local evidence but only partial rooted-tree evidence.

6.2. Local Features Need Rooted BFS Relations

To understand which information can actually solve the SPD task, we train shallow logistic-linear classifiers on explicit graph features to understand which SPD classes become solvable if an algorithm has access to different features.

Table 6 shows local features and short-distance features are enough for distances 1 and 2, but not sufficient for the full task. The local feature set (A) reaches only 0.684 overall and 0.315 on distance 4. **Adding BFS-tree pair relations**

[9] Here, $\mathcal{N}(i)$ denotes the set of nodes that are connected to node i, and $\deg(i)$ denotes the degree of node i.

[10] We identify that the model tends to choose the highest closeness centrality node for 65% of the test graphs. However, we do not have conclusive empirical evidence to pinpoint its selection criteria.

Table 6. Explicit feature experiments for SPD. Each row uses (A) local features, (B) A + pair relations from BFS tree, (C) BFS-parent degree/child-count, and (D) all features (A + B + C). See Appendix J for definitions. Class n is abbreviated to Cn.

	Overall	C0 d=1	C1 d=2	C2 d=3	C3 d=4	C4 $d\geq5$
(A)	0.684	1.000	1.000	0.561	0.315	0.762
(B)	**0.899**	1.000	0.999	0.821	0.824	0.928
(C)	0.414	0.045	0.479	0.333	0.266	0.698
(D)	0.903	1.000	0.999	0.825	0.835	0.930

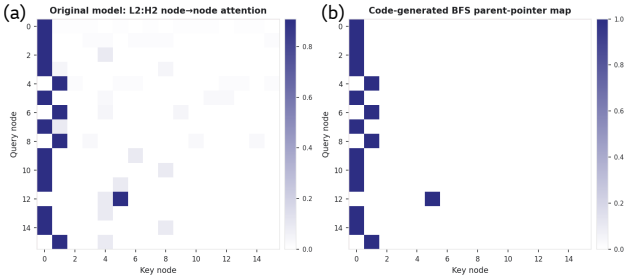

Figure 4. Visualization of the L2:H2 attention intervention experiment. (a) Learned L2:H2 node-to-node attention map on one held-out shortest-path graph. (b) The diagnostic BFS-parent source map generated by the replacement code for the same graph.

Table 7. L2:H2 attention intervention experiment. In Top-1 only, we collapse the attention into a one-hot attention with the highest attention. In BFS-parent, we collapse to the the closeness-root BFS parent. As a baseline, we test choosing a random neighbor of v or constructing a uniform attention weight over neighbor nodes.

	Overall	C2 d=3	C3 d=4	C4 $d\geq5$
Original	0.989	0.995	0.984	0.981
Top-1 only	0.989	0.996	0.982	0.979
BFS-parent	0.983	0.986	0.971	0.977
Random neigh.	0.821	0.730	0.567	0.977
Uniform neigh.	0.824	0.771	0.477	0.982

Table 8. Probe accuracies at the L2 z-vector at H2 (*i.e.*, $z^{(2,2)}$). The probe asks whether the head output is more predictive of the query node or the copied source node.

Feature	Query v	Source $a(v)$	Δ
Degree	0.435	0.760	+0.325
Leaf neigh. count	0.760	0.519	−0.241
High-degree neigh. count	0.831	0.637	−0.194
BFS depth	0.503	0.534	+0.031
BFS child count	0.545	0.579	+0.034

raises accuracy to 0.899, close to the full explicit feature set at 0.903. Thus, the model needs variables such as BFS-tree distance, same-parent, parent-child, same-depth, and same-first-hop-branch to turn local summaries into pairwise distance evidence. We next show that L2:H2 helps move parent-side information into each node.

6.3. L2:H2 Acts as a BFS-Parent Copy Head

We found that the central L2 component is L2:H2, which we call it a **BFS-parent copy head**. L2:H2 has two functional properties. First, it recognizes the *BFS-parent* node. Specifically, for a given query node v, its attention weights often focuses on a node $a(v)$ that is a direct neighbor (*i.e.*, $a(v) \in \mathcal{N}(v)$) and simultaneously a parent (*i.e.*, predecessor) of a rooted BFS tree. Second, it is a *copy* head, where it selectively writes the source node's value into the query-node. This operations makes L2:H2 the most causally active head in L2, where removing L2:H2 drops SPD accuracy from approximately 0.988 to 0.611, comparable to ablating the entire L3 attention layer. Ablating the other L2 heads does not reduce accuracy below 0.980. Overall, this indicates that later layers heavily rely on L2:H2. We now describe various empirical evidence for these claims.

L2:H2 often selects a BFS-parent. Firstly, we tackle the attention pattern of L2:H2. Empirically, we find that (1) the top-1 selected $a(v)$ is adjacent (*i.e.*, is a neighbor node) with rate 1.000 with an average of 95% of the node token attention mass being allocated to it, while also being the closeness-root BFS parent 81.2% of the time.

Table 7 shows a clear behavior of the attention map in L2:H2. Collapsing the learned attention to its top-1 source leaves accuracy unchanged, which shows the attention weights were already highly concentrated. Replacing the source with the closeness-root BFS parent (see Figure 4) also preserves high accuracy to 0.983, strongly indicating a BFS-parent selection behavior. As a contrast, the random and uniform neighbor source baselines both fail on class 2 and 3.

Selective copying the BFS-parent representation. We now test what L2:H2 copies from the source node. For each query node v, we take the copied source $a(v)$, extract $z^{(2,2)}$, and train linear probes to predict either the query node's feature or the source node's feature.

Table 8 shows that the source node's degree information is most clearly being copied, resulting in more than 0.3 accuracy difference compared to the query node. A direct value-space comparison gives the same conclusion. The L2:H2 output has cosine similarity 0.994 with the source node's value vector, compared with 0.091 for the query node's value vector. BFS depth and child count show weak source node copying behavior. In contrast, leaf-neighbor and high-degree-neighbor information does not exhibit copying behavior.

Table 9. Cosine similarity between dominant right-singular directions of the class-specific SPD projections W_c. Blue cells mark high similarity, *i.e.*, alignment. Class n is abbreviated to Cn.

	C0 d=1	C1 d=2	C2 d=3	C3 d=4	C4 d≥5
C0	–	0.487	0.046	0.023	0.018
C1	0.487	–	0.012	0.054	0.097
C2	0.046	0.012	–	0.959	0.891
C3	0.023	0.054	0.959	–	0.966
C4	0.018	0.097	0.891	0.966	–

6.4. A Shared Axis for Longer Distances

After L2:H2, the 32-dimensional $z_i^{(2,2)}$ vector seems to contain *partial* depth-like structure. A linear probe on $z_i^{(2,2)}$ predicts BFS depth from the inferred root with 0.503 accuracy. However, this representation still requires further refinement by the model. Now, we attempt to analyze how the model arrives at its conclusion after L2.

To this end, we analyze the SPD task head first before analyzing the effect of L3. For context, the model computes class-specific bilinear logits $\mathrm{logit}_c(i,j) = (W_c r_i^\star)^\top (W_c r_j^\star)$, where $r_i^\star$ is the final residual vector of node i, and c indexes the SPD output class. Each $W_c \in \mathbb{R}^{32\times 256}$ defines a class-specific projection before the pairwise dot product. We therefore take the top right-singular vector of each W_c, which gives the dominant residual-stream direction read by that class projection.[11]

Table 9 shows that the task head separates short-distance and longer-distance readouts. Class 0 and 1 (*i.e.*, distances 1 and 2) are moderately aligned with each other (∼0.5) but nearly orthogonal to class 2–4 (*i.e.*, distances more than 2). In contrast, class 2, 3, and 4 have highly aligned top directions, with pairwise cosine 0.891–0.966. This suggests that the final task head reuses similar distance-like residual stream directions for longer-distance classes, while using different ones for closer node pairs.

6.5. L3 Refines Longer Distance Representations

Since early local feature computations can gather enough evidence for shorter node pairs and L2 performs a copy operation along the BFS tree, we hypothesize L3 is responsible for refining the representation for more distance node pairs. To start with, we identify that L3 benefits SPD classes with longer distances by a simple experiment, where we zero-ablate the L3 attention output and measure accuracy separately by true graph distance. Table 10 shows that L3 attention has little effect on nearby node pairs, and while the performance drops significantly for longer distances.

To dive in more deeply, we collect the top right-singular vector of class-specific heads W_2, W_3, and W_4, denoted as s_2, s_3, and s_4, respectively. Then, we define the shared long-distance direction $\hat{w}_{\text{dist}}^{(3)}$ as the first principal component of $\{s_2, s_3, s_4\}$. For each node pair (i,j) at some residual stream position τ, we compute a symmetric pairwise score $p_{ij}^{(\tau)} = \left\langle r_i^{(\tau)}, \hat{w}_{\text{dist}}^{(3)} \right\rangle + \left\langle r_j^{(\tau)}, \hat{w}_{\text{dist}}^{(3)} \right\rangle$ to capture how much the representation is aligned with $\hat{w}_{\text{dist}}^{(3)}$ and their correlation with the actual distance. Table 11 shows that this projection is already correlated with distance at L2-post and becomes more separated by L3-post. At L2-post, the projection sum spans from -3.65 at distance $d = 2$ to $+1.39$ at $d \geq 5$. At L3-post, it moves from -10.52 at $d = 2$ to -0.75 at $d \geq 5$. To summarize, L3 primarily refines longer-distance pair representations for the bilinear readout, analogous to the MLP refinement in Section 4.1.

Table 10. Per-distance SPD accuracy after zero-ablating the L3 attention output. Distances are true graph shortest-path distances.

SPD	Clean	L3-attn ablated	Δ acc.
1 (adjacent)	1.000	0.987	−0.013
2	0.999	0.869	−0.130
3	0.996	0.880	−0.116
4	0.990	0.605	**−0.385**
5+	0.985	0.299	**−0.686**

Table 11. Pairwise projection sum onto the shared long-distance direction $\hat{w}_{\text{dist}}^{(3)}$. Distances are true graph shortest-path distances.

SPD	# pairs	L2-post projection	L3-post projection
2	2,586	-3.65 ± 3.93	-10.52 ± 7.80
3	3,603	-2.49 ± 3.33	-8.90 ± 6.08
4	3,472	-0.43 ± 3.22	-5.00 ± 5.32
5+	2,158	$+1.39 \pm 2.96$	-0.75 ± 4.56

7. QK Circuit Analysis and ID Matching

All analyses in previous sections so far are based on the model's capability to attend to the correct edge tokens by matching the node ID vectors e_i. As the final analysis, we now ask *how* this ID matching is made possible. In the TokenGT-style input, the model does not have architectural constraints like GNNs. It must instead recover node incidence from the given IDs contained in node and edge tokens.[12] We find that **L0 query-key (QK) weights universally learn node-incidence tests implemented through ID matching** (see Appendix H for more discussions).

This node incidence is the local relation required for permutation-equivariant computation. Specifically, for any edge (u,v) and node permutation $\pi : V \to V$, if $i \in \{u,v\}$ then $\pi(i) \in \{\pi(u), \pi(v)\}$ and vice versa. This relation can

[11] We refer to Appendix G for a short primer on singular vectors.

[12] Here, node incidence means locating edge tokens (u,v) such that the query node ID i satisfies $i = u$ or $i = v$.

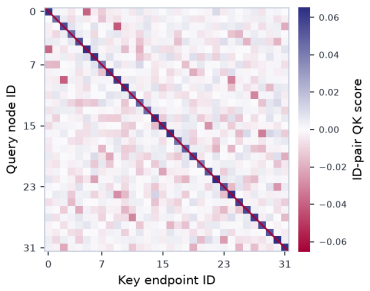

Figure 5. $M_{vu}^{(5)}$ of L0:H5 from the SPD model. Rows are node-query identifier labels and columns are edge-key endpoint labels. The strong diagonal shows that the head gives larger QK scores when the node ID matches the edge endpoint ID.

be tested from tokens by checking whether a node identifier e_i equals *either* endpoint ID vectors e_u or e_v. Naturally, a similar equality is also preserved: if $e_i = e_u$ then $e_{\pi(i)} = e_{\pi(u)}$ and vice versa. Therefore, if the QK circuit is driven by ID equality, the incident-edge attention pattern should dynamically reconfigure according to the permutation π. This is the operation needed for permutation-consistent incident-edge retrieval.

Node-incidence tests are distributed across L0. Consider the node-to-edge attention in some L0 layer. A node query for node i is $[\tau_{\text{node}} \| e_i \| e_i]$, while an edge key for edge (u, v) is $[\tau_{\text{edge}} \| e_u(\text{'}u\text{ slot'}) \| e_v(\text{'}v\text{ slot'})]$. For a head to retrieve incident edges, it must give a higher QK score when $e_i = e_u$ or $e_i = e_v$. We measure if this equality test exists in the weight space.

Write the input projection as $W_{\text{proj}} = [W_{\text{type}} \mid W_u \mid W_v]$, where W_u and W_v project the first and second ID positions in the token. As an example, take the node query's second ID position and the edge key's first ID position. Their composed maps into head h's query and key spaces are

$$C_Q^{(v,h)} = W_Q^{(0,h)} W_v, \qquad C_K^{(u,h)} = W_K^{(0,h)} W_u.$$

The ID-only QK contribution from node ID i to edge endpoint ID j through this position pair is described through the slot-pair QK matrix

$$M_{vu}^{(h)}[i, j] = e_i^\top (C_Q^{(v,h)})^\top C_K^{(u,h)} e_j.$$

If $M_{vu}^{(h)}$ is diagonally dominant, then this part of the head gives a larger QK score when $i = j$; in the node-to-edge case, this detects the $i = u$ endpoint match. The other three variants $M_{uu}^{(h)}, M_{uv}^{(h)}, M_{vv}^{(h)}$ are defined analogously. Since the node token contains e_i in both ID positions, any diagonally dominant $M_{xu}^{(h)}$ can detect $i = u$, and any diagonally dominant $M_{xv}^{(h)}$ can detect $i = v$.

We quantify diagonal dominance by *selectivity*, $\text{Sel}(M)$: the average same-ID QK score advantage over different-ID scores, measured in off-diagonal standard deviations.[13] In Table 12, we observe that all eight L0 heads include at least one $M_{xy}^{(h)}$ that exceeds 3σ selectivity in DC and RM. In SPD, seven out of eight L0 heads exceed 3σ, with the remaining head still positive at 2.70σ, empirically indicating the universality of this feature.[14] A runtime check using the test set gives the same conclusion: every L0 head assigns higher pre-softmax scores to incident edge tokens than to nonincident edge tokens in all three models.

Table 12. ID matching is distributed across L0. In the second column, we count the number of L0 heads whose best identifier field pair has $\text{Sel}(M) > 3$. The third column reports the strongest node-to-incident-edge pre-softmax gaps.

Task	$\text{Sel}(M) > 3$?	Strongest runtime heads
DC	8/8	H2 2.31σ, H4 2.17σ, H5 1.95σ
RM	8/8	H4 2.27σ, H2 1.92σ, H7 1.59σ
SPD	7/8	H5 2.34σ, H0 2.15σ, H7 1.84σ

We also visualize $M_{vu}^{(5)}$ in the L0:H5 of the SPD model ($\text{Sel}(M) = 8.02\sigma$) in Figure 5. This channel compares the node query's v-slot identifier to the edge key's u-slot identifier, so the diagonal corresponds to one side of the incident-edge test. We also provide visualizations when the model uses an expanded set of 64 node ID vectors in Appendix I.2, where it also shows a strong diagonal pattern despite operating on $d_{\text{kv}} = 32$-dimensional QK space. This is possible because the model only needs same-ID scores to maintain a sufficient margin over different-ID scores, rather than realizing a strict identity matrix.

8. Limitations and Final Discussions

This work has several limitations. First, our experiments only used a vanilla transformer with TokenGT-like input with relatively small synthetic graphs. Extension to larger graphs, richer graph features, different tokenizations, and other models are therefore an important future direction. Second, a full identification of the fine-grained circuit for the RM model is incomplete. Third, the QK analysis does not fully explain how these matrices emerge during training and their robustness in larger node-ID vocabularies.

Despite these limitations, our results provide one of the first mechanistic interpretation of how a vanilla transformer computes node and edge tokens. In short, the model encodes local graph structure through ID-based incident-edge retrieval, then composes this information in task-specific ways. We hope this work provides a useful starting point for extending mechanistic interpretability towards graphs.

[13] For example, selectivity of 2σ means that same-ID QK scores are, on average, two off-diagonal standard deviations higher than different-ID scores. See Appendix I.1 for the technical definition.

[14] See Appendix I.3 for a comprehensive result.

Acknowledgements

We would like to thank SungWoo Hur for the early discussions regarding training tokenized graph transformers during his time as an intern in LG AI Research. We also like to thank the anonymous reviewers that helped improve this work, especially regarding the QK circuit analysis.

Impact Statement

This paper presents work whose goal is to expand the field of mechanistic interpretability outside natural language inputs, and specifically towards graph inputs. We expect unveiling the inner workings of graph transformers will advance the field towards a ubiquitous understanding of deep learning models. Since this is an early stage of research, we do not see any immediate foreseeable societal consequences of our work.

References

Bereska, L. and Gavves, S. Mechanistic interpretability for AI safety - A review. *Trans. Mach. Learn. Res.*, 2024, 2024.

Bibal, A., Cardon, R., Alfter, D., Wilkens, R., Wang, X., François, T., and Watrin, P. Is attention explanation? an introduction to the debate. In *ACL*, pp. 3889–3900, Dublin, Ireland, May 2022.

Bronstein, M. M., Bruna, J., Cohen, T., and Velickovic, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478, 2021.

El, B., Choudhury, D., Liò, P., and Joshi, C. K. Towards mechanistic interpretability of graph transformers via attention graphs. *CoRR*, abs/2502.12352, 2025.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, pp. 1263–1272, Sydney, NSW, Aug. 2017.

Kim, J., Nguyen, D., Min, S., Cho, S., Lee, M., Lee, H., and Hong, S. Pure transformers are powerful graph learners. In *NeurIPS*, New Orleans, LA, Nov.–Dec. 2022.

Lieberum, T., Rajamanoharan, S., Conmy, A., Smith, L., Sonnerat, N., Varma, V., Kramár, J., Dragan, A. D., Shah, R., and Nanda, N. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *CoRR*, abs/2408.05147, 2024.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *ICLR*, New Orleans, LA, May 2019.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in GPT. In *NeruIPS*, New Orleans, LA, Nov.–Dec. 2022.

Müller, L., Galkin, M., Morris, C., and Rampásek, L. Attending to graph transformers. *Trans. Mach. Learn. Res.*, 2024, 2024.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

Sanford, C., Fatemi, B., Hall, E., Tsitsulin, A., Kazemi, M., Halcrow, J., Perozzi, B., and Mirrokni, V. Understanding transformer reasoning capabilities via graph algorithms. In *NeurIPS*, BC, Canada, Dec. 2024.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021.

Yan, Z., Ma, T., Gao, L., Tang, Z., Chen, C., and Wang, Y. Cycle invariant positional encoding for graph representation learning. In *LoG*, Virtual event, Nov. 2023.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T. Do transformers really perform badly for graph representation? In *NeurIPS*, Virtual event, Dec. 2021.

Zhang, F. and Nanda, N. Towards best practices of activation patching in language models: Metrics and methods. In *ICLR*, Vienna, Austria, May 2024.

A. Residual stream and probing/ablation points

Table 13. Probing and intervention points with definitions.

Symbol	Definition
$r_{\text{pre}}^{(0)}$	Token embedding after W_{proj}
$r_{\text{pre}}^{(\ell)}$	Input to layer ℓ; for $\ell > 0$, $r_{\text{pre}}^{(\ell)} = r_{\text{post}}^{(\ell-1)}$
$r_{\text{mid}}^{(\ell)}$	$r_{\text{pre}}^{(\ell)} + a^{(\ell)}$
$r_{\text{post}}^{(\ell)}$	$r_{\text{mid}}^{(\ell)} + m^{(\ell)}$
$z_i^{(\ell,h)}$	Output of head h at token i before $W_O^{(\ell,h)}$
$\alpha_{ij}^{(\ell,h)}$	Attention weight from query token i to key/value token j
$q_i^{(\ell,h)}, k_j^{(\ell,h)}, v_j^{(\ell,h)}$	Query, key, and value vectors
$a^{(\ell)}$	Self-attention output before residual addition
$m^{(\ell)}$	MLP output before residual addition

Each layer $\ell \in \{0, 1, 2, 3\}$ applies a self-attention sublayer followed by an MLP sublayer, both with residual connections. The forward pass through layer ℓ is

$$
\begin{aligned}
a^{(\ell)} &= \mathrm{SelfAttn}^{(\ell)}\left(\mathrm{LN}(r_{\text{pre}}^{(\ell)})\right), \\
r_{\text{mid}}^{(\ell)} &= r_{\text{pre}}^{(\ell)} + a^{(\ell)}, \\
m^{(\ell)} &= \mathrm{MLP}^{(\ell)}\left(\mathrm{LN}(r_{\text{mid}}^{(\ell)})\right), \\
r_{\text{post}}^{(\ell)} &= r_{\text{mid}}^{(\ell)} + m^{(\ell)}.
\end{aligned}
$$

Here, LN denotes RMSNorm. The self-attention output decomposes into per-head contributions:

$$
\begin{aligned}
a^{(\ell)} &= \sum_{h=0}^{7} W_O^{(\ell,h)} z^{(\ell,h)}, \\
z_i^{(\ell,h)} &= \sum_j \alpha_{ij}^{(\ell,h)} W_V^{(\ell,h)} \, \mathrm{LN}\left(r_{\text{pre},j}^{(\ell)}\right).
\end{aligned}
$$

In later sections, "post-embed" refers to $r_{\text{pre}}^{(0)}$, "post-Lℓ attn" to $r_{\text{mid}}^{(\ell)}$, and "post-Lℓ MLP" to $r_{\text{post}}^{(\ell)}$.

B. Full Layer-wise Ablation Results for DC

Table 14. The full layer-wise ablation results for the degree counting task. The table shows the model's performance after zero-ablating specific parts of the input.

Ablated component	Accuracy after ablation	Δ_{task}	Interpretation
L0 attention output	0.3902	−0.6098	Primary incident-edge readout
L0 MLP output	0.8262	−0.1738	Separates the L0 degree feature
L1 attention output	0.9040	−0.0960	Small backup/refinement
L1 MLP output	0.8537	−0.1463	Small backup/refinement
L2 attention output	0.9992	−0.0008	Negligible
L2 MLP output	0.9878	−0.0122	Negligible
L3 attention output	0.9825	−0.0175	Negligible
L3 MLP output	0.9848	−0.0152	Negligible

C. Per Head Analysis of Attention Concentration in Incident Edge Tokens

Refer to Table 15.

D. Comparison of L0:H5's Incident-edge Attention Mass

Refer to Table 16.

Table 15. Incident vs nonincident per-position attention ratio per L0 head (50 graphs, node-degree model).

H0	H1	H2	H3	H4	H5	H6	H7
7.38×	17.72×	8.28×	17.82×	5.85×	8.11×	7.77×	7.32×

Table 16. Comparison of the L0:H5's incident-edge attention mass against other attention mass, varied by true node degree. The results directly show that the attention increases its total attention mass to incident edges, likely since it simply has more edges to attend to.

Degree	**Inc. edge mass**	Self mass	Other mass
1	0.044	0.184	0.772
2	0.190	0.158	0.653
3	0.338	0.132	0.530
4	0.485	0.103	0.412
5	0.563	0.083	0.354

E. Validation: Explicit Algorithm Test

Table 17. Explicit leaf-neighbor baselines versus the learned ring classifier.

Algorithm	Accuracy
Majority baseline (predict ring)	0.502
deg=1 → non-ring; else → ring	0.852
deg≥2 AND no leaf neighbor → ring; else non-ring	0.839
Same with best single threshold (frac $\leq$ 0.4)	0.939
Same with per-degree optimal thresholds	0.954
Neural network	**1.000**

To validate that the described mechanism is the core computation, we implement it as a hand-coded algorithm and measure accuracy on the full test set (12,500 graphs, 327k nodes):

```
def predict_ring(v, adj, degree, threshold):
    """Explicit leaf-fraction algorithm (best single-threshold variant)."""
    if degree[v] < 2:
        return 0  # non-ring: leaves cannot be on a cycle
    frac_leaf = sum(1 for u in adj[v] if degree[u] == 1) / degree[v]
    return 1 if frac_leaf <= threshold else 0  # ring if few/no leaf neighbors
```

The leaf-neighbor-fraction algorithm achieves 95.4% with optimally tuned per-degree thresholds, indicating that it is the dominant feature but not the complete story. The model's 100% accuracy likely requires the ensemble of 8 redundant L1 heads (each contributing slightly different neighborhood features from their respective W_V projections) plus later-layer refinement in L2–L3. The residual 5% gap likely corresponds primarily to ring nodes with degree ≥ 3 that have partial leaf neighbors (tree branches hanging off the ring) — cases where the fraction feature overlaps between classes and finer-grained discrimination is needed. Nevertheless, further investigation is required to fully back this claim.

F. Additional Discussions

F.1. Structured Redundancies

For the DC task, we have noticed that several heads produce highly structural attention patterns for causally inactive (*i.e.*, zero-ablation only drops the model performance $\sim 1\%$). These results demonstrate that causal validation is necessary (such as targeted interventions) and observing attention patterns is not enough, which also aligns with the previous consensus that attention is a weak evidence in terms of model explanation (Bibal et al., 2022).

However, the question remains: What makes the model to train structured redundancies and are there any benefits in such? At this point, we do not have any empirical evidence but point to several potential avenues:

- **Artifacts of early training.** Since all model parameters are randomly initialized at the beginning of the training process, it is possible that such structured model weights are simply artifacts of that era. In this case, the model must

have learned that the initial layer is sufficient to solve DC sometime during the training phase and mainly focused on layer 0 afterwards.

- **Better gradients.** During training, since the gradients are affected by the previous (i.e., closer to the output) layer's weight and activations, it is possible that the structure of later layers are mainly used to channel gradient information to target specific parts in the early layers.

G. A Short Primer on Singular Vectors

Many analyses in this paper study a learned linear map

$$W \in \mathbb{R}^{m \times d},$$

which takes a residual-stream vector $r \in \mathbb{R}^d$ and maps it to a lower-dimensional feature vector $Wr \in \mathbb{R}^m$. A useful question is: which directions in the input space does W read most strongly?

The singular value decomposition (SVD) writes

$$W = U \Sigma V^\top,$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{d \times d}$ are orthonormal matrices, and Σ contains nonnegative singular values

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq 0.$$

The columns of V, denoted $v_1, v_2, \ldots$, are the *right-singular vectors*. These are directions in the input space of W. In our setting, this input space is usually the residual stream. In this setup, the *top right-singular vector* v_1 is the unit input direction that W amplifies most:

$$v_1 = \arg \max_{\|r\|_2 = 1} \|Wr\|_2.$$

The corresponding maximum value is the top singular value:

$$\|Wv_1\|_2 = \sigma_1.$$

Thus, v_1 can be interpreted as the strongest single input direction read by W (Note that, this should not be read as v_1 being the only direction that is being used).

More generally, the first k right-singular vectors span the k-dimensional input subspace to which W is most sensitive. If most of the squared singular-value mass is concentrated in the first few singular values, then W is effectively low-rank, *i.e.*, it mostly reads a small number of input directions. If the singular values decay slowly, then W reads a broader subspace.

H. Incidence Preservation and Permutation Symmetry

Here, we describe how incidence preservation supports permutation symmetry, extending the discussion in the introduction and Section 7.

Incidence preservation and graph symmetry. Let $G = (V, E)$ be a graph and let $\pi : V \to V$ be a permutation of node labels (*i.e.*, one specific way to relabel the nodes). Here, each node v from the original graph G is relabeled to $\pi(v)$ in the relabeled graph. Then, relabeled graph is denoted as $\pi \cdot G = (V, \pi(E))$, where we slightly abuse the notation of π that can act on edges (for an edge $e = (u, v)$, $\pi(e) = (\pi(u), \pi(v))$) and set of edges (for a set of edges E, $\pi(E) = \{(\pi(u), \pi(v)) : (u, v) \in E\}$). For an edge $e = (u, v)$, define the incidence relation

$$R_e(i) = \mathbf{1}\{i = u \vee i = v\}.$$

Relabeling preserves this relation:

$$R_e(i) = R_{\pi(e)}(\pi(i)).$$

Note that this statement is not in itself the definition of permutation equivariance or invariance. However, it is a practical property which a model can exploit to achieve such permutation symmetry without depending on arbitrary node labels.

For some function f, permutation equivariance means

$$f(\pi \cdot G) = \pi \cdot f(G),$$

where the output attached to node i moves to node $\pi(i)$ (*i.e.*, the output follows the same permutation π). Also, permutation invariance means

$$F(\pi \cdot G) = F(G),$$

where the output is unaffected by π.

A nontrivial model that utilizes graph connectivity while preserving permutation symmetry must use connectivity through relations that are *stable under relabeling* (a trivial constant model can still satisfy such constraints without using edges). In our case with TokenGT-style models, **incidence** is the basic relation.

How GNNs achieve permutation symmetry. A message passing graph neural network (GNN) layer updates the representation of node i by

$$h_i' = \phi\left(h_i,\ \mathrm{AGG}_{j:(i,j)\in E}\, \psi(h_i, h_j, e_{ij})\right),$$

where ϕ and ψ are parameterized functions and AGG is permutation-invariant, such as sum or mean (Bronstein et al., 2021). A key point is that the aggregation index is **defined** by graph connectivity: node i aggregates only from nodes j such that $(i, j) \in E$.

Under a node relabeling π, the neighbor set transforms as

$$\mathcal{N}(i) = \{j : (i, j) \in E\} \quad \mapsto \quad \mathcal{N}_{\pi\cdot G}(\pi(i)) = \{\pi(j) : j \in \mathcal{N}(i)\}.$$

Thus the same multiset of messages is attached to the relabeled node $\pi(i)$, which is natural as we did not change the graph structure and only changed the node label. Assuming node and edge features are relabeled consistently, the updated representation satisfies

$$h'_{\pi(i)}(\pi \cdot G) = h'_i(G),$$

up to the corresponding relabeling of node-indexed outputs. Therefore the layer is permutation-equivariant. Stacking such layers preserves equivariance. A permutation-invariant readout, such as

$$F(G) = \rho\left(\sum_{i\in V} h_i\right),$$

then gives graph-level invariance:

$$F(\pi \cdot G) = \rho\left(\sum_{i\in V} h_{\pi(i)}\right) = \rho\left(\sum_{i\in V} h_i\right) = F(G).$$

Relation to identifier matching in TokenGT-style transformers. As seen from the description above, GNNs use *incidence by construction*: their aggregation is indexed by the edge set. By contrast, transformers does not have this restriction, *i.e.*, node token can attend to all node and edge tokens, so it must *learn* which edge tokens are incident to it.

Assuming the TokenGT-style input, a node token for node i contains $[e_i, e_i]$, while an edge token (u, v) contains $[e_u, e_v]$, where u and v are the endpoint nodes of the edge. Therefore incidence can be recovered by ID matching:

$$i \in \{u, v\} \quad \Longleftrightarrow \quad (e_i = e_u) \text{ or } (e_i = e_v).$$

Because the identifier assignment is one-to-one, equality is preserved under relabeling:

$$e_i = e_u \quad \Longleftrightarrow \quad e_{\pi(i)} = e_{\pi(u)}.$$

Therefore, if the L0 QK attention assigns high scores when a node identifier matches an edge endpoint identifier, the selected edge tokens relabel consistently with π. This does not guarantee nor prove that the full transformer is exactly equivariant or invariant: this is a learned behavior and may break down on other distributions and later computation may use other features.

I. Further Details on QK Circuit Analysis

I.1. Technical Definitions.

Selectivity. For a matching matrix $M \in \mathbb{R}^{K\times K}$, we define

$$\mathrm{Sel}(M) = \frac{\frac{1}{K}\sum_i M_{ii} - \frac{1}{K(K-1)}\sum_{i\neq j} M_{ij}}{\mathrm{Std}_{i\neq j}(M_{ij})}.$$

Runtime logit gap. For node token i, edge token e, and head h, let

$$s_{i,e}^{(h)} = \frac{(q_i^{(0,h)})^\top k_e^{(0,h)}}{\sqrt{d_{\mathrm{kv}}}}$$

be the node-to-edge pre-softmax score. With $\mathcal{I} = \{(i,e) : i \in e\}$ and $\mathcal{N} = \{(i,e) : i \notin e\}$, we report

$$\Delta_{\mathrm{runtime}}^{(h)} = \frac{\mathbb{E}_{\mathcal{I}} s_{i,e}^{(h)} - \mathbb{E}_{\mathcal{N}} s_{i,e}^{(h)}}{\sqrt{\frac{1}{2}\left(\mathrm{Var}_{\mathcal{I}}(s_{i,e}^{(h)}) + \mathrm{Var}_{\mathcal{N}}(s_{i,e}^{(h)})\right)}}.$$

This is the incident-vs.-nonincident gap in units of pooled standard deviation.

Worked example. For an edge token $(u,v) = (3,7)$ and a node query token for $i = 7$, the identifier-only QK score contains

$$\left\langle C_Q^{(u,h)} e_7 + C_Q^{(v,h)} e_7,\ C_K^{(u,h)} e_3 + C_K^{(v,h)} e_7 \right\rangle$$
$$= M_{uu}^{(h)}[7,3] + M_{uv}^{(h)}[7,7] + M_{vu}^{(h)}[7,3] + M_{vv}^{(h)}[7,7].$$

The diagonal terms receive the same-identifier bonus. For a nonincident edge such as $(3,11)$, identifier 7 hits no diagonal entry.

I.2. ID-pair scoring matrix using 64 node IDs

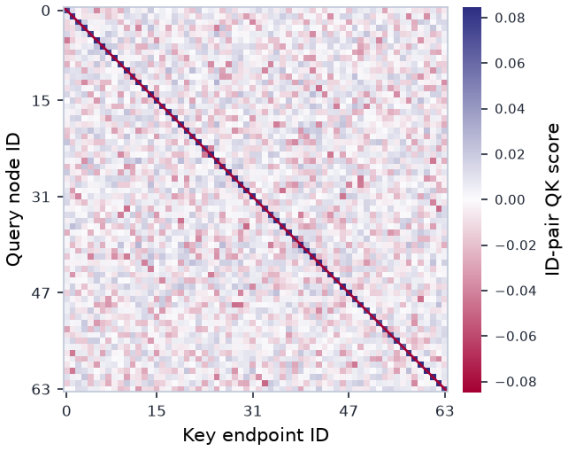

Figure 6. $M_{vu}^{(5)}$ of L0:H5 from the SPD model, trained with all 64 node IDs, exceeding the ($d_{kv} = 32$) QK dimension.

I.3. Comprehensive slot-pair scan.

The main text compresses each head to its strongest identifier field pair. The uncompressed scan evaluates all four slot pairs,

$$\left\{\mathrm{Sel}(M_{uu}^{(h)}), \mathrm{Sel}(M_{uv}^{(h)}), \mathrm{Sel}(M_{vu}^{(h)}), \mathrm{Sel}(M_{vv}^{(h)})\right\},$$

for each L0 head $h \in \{0, \ldots, 7\}$ and each task model, giving $4 \times 8 \times 3 = 96$ selectivity values. The head-level count in Table 12 is obtained by taking

$$\max_{x,y \in \{u,v\}} \mathrm{Sel}(M_{xy}^{(h)})$$

for each head and checking whether this maximum exceeds 3σ. Thus the statement $8/8$ does not mean that all four slot-pair matrices are strongly diagonal for every head; it means that every head has at least one strong identifier field pair.

Table 18. Comprehensive active-label identifier matching scan. Each row reports selectivity for all four ID-pair matrices of one L0 head. Bold marks the strongest identifier field pair used by the main-text head count.

Task	Head	M_{uu}	M_{uv}	M_{vu}	M_{vv}
DC	H0	**7.5**	2.1	6.7	0.6
DC	H1	-3.1	4.2	2.6	**7.9**
DC	H2	**7.9**	4.3	6.7	4.3
DC	H3	-3.3	4.3	3.1	**8.0**
DC	H4	**7.2**	4.5	5.6	4.4
DC	H5	**7.5**	3.8	7.1	2.5
DC	H6	**7.5**	1.9	6.8	0.0
DC	H7	**6.6**	4.8	5.7	4.5
RM	H0	**7.7**	1.1	3.8	-3.8
RM	H1	-6.7	2.5	-1.4	**6.8**
RM	H2	**7.6**	3.9	3.4	-1.0
RM	H3	-9.2	1.9	-2.3	**6.3**
RM	H4	**5.9**	4.3	3.2	3.2
RM	H5	4.4	1.1	**9.6**	-0.1
RM	H6	**7.8**	-0.5	5.2	-4.4
RM	H7	2.9	4.5	2.5	**6.5**
SPD	H0	2.2	1.7	**8.1**	6.3
SPD	H1	**2.7**	1.0	2.6	-1.4
SPD	H2	**7.2**	5.1	-4.0	-5.3
SPD	H3	-5.6	-3.5	2.6	**6.0**
SPD	H4	**6.6**	4.0	-2.9	-4.9
SPD	H5	1.6	1.9	**8.0**	7.1
SPD	H6	**7.3**	-0.6	6.8	-2.7
SPD	H7	5.1	2.1	**7.2**	1.9

Table 18 shows this comprehensive scan. The strongest identifier field pair can differ by head. M_{uu} and M_{vv} match the same identifier slot across query and key, while M_{uv} and M_{vu} match across token slots. All four cases can support endpoint matching because node tokens contain e_i in both identifier slots and edge tokens contain e_u, e_v.

J. Explicit Feature Definitions for Table 6

his appendix defines the feature sets used in Table 6. These are hand-computed graph features used only for the explicit-feature experiments. They are not given to the transformer during training or evaluation.

Let $G = (V, E)$ be the input graph, and let $N(v)$ be the set of neighbors of node v. For a node v, define

$$\deg(v) = |N(v)|, \qquad \ell(v) = |\{u \in N(v) : \deg(u) \leq 1\}|, \qquad h(v) = |\{u \in N(v) : \deg(u) \geq 4\}|.$$

Here $\ell(v)$ is the leaf-neighbor count and $h(v)$ is the high-degree-neighbor count. For a node pair (i, j), define the direct short-distance features

$$A_{ij} = \mathbf{1}[(i, j) \in E], \qquad C_{ij} = |N(i) \cap N(j)|.$$

$A_{ij} = 1$ certifies distance 1, and $C_{ij} > 0$ certifies distance at most 2.

For the rooted features, we choose the root r as the node with maximum closeness centrality and construct a BFS tree from r. Let $p_r(v)$ be the selected BFS parent of v, with $p_r(r) = r$. Let $d_r(v)$ be BFS depth, $b_r(v)$ be the first child of r on

the selected BFS-tree path from r to v, and $c_r(v) = |\{u : p_r(u) = v\}|$ be the BFS child count. We define $D_r(i, j)$ as the number of tree edges on the selected BFS-tree path between i and j.

The four feature sets in Table 6 are:

$$\begin{aligned} x_{ij}^{(A)} = [&A_{ij}, C_{ij}, \deg(i) + \deg(j), |\deg(i) - \deg(j)|, \\ &\ell(i) + \ell(j), |\ell(i) - \ell(j)|, h(i) + h(j), |h(i) - h(j)|]. \end{aligned}$$

This is (A), the local feature set. It contains adjacency, common-neighbor count, and local count features computed from the queried nodes i and j.

$$\begin{aligned} x_{ij}^{(B)} = [&x_{ij}^{(A)}, D_r(i, j), \mathbf{1}[p_r(i) = p_r(j)], \mathbf{1}[p_r(i) = j \vee p_r(j) = i], \\ &\mathbf{1}[d_r(i) = d_r(j)], \mathbf{1}[b_r(i) = b_r(j)], d_r(i) + d_r(j), |d_r(i) - d_r(j)|, \\ &c_r(i) + c_r(j), |c_r(i) - c_r(j)|]. \end{aligned}$$

This is (B), A + pair relations from the BFS tree. It contains all columns in (A) and then adds relations between the queried nodes i and j in the closeness-root BFS tree. It does not include degree or child-count features of the BFS parents.

$$x_{ij}^{(C)} = [\deg(q) + \deg(t), |\deg(q) - \deg(t)|, c_r(q) + c_r(t)], \qquad q = p_r(i), \quad t = p_r(j).$$

This is (C), the BFS-parent degree/child-count feature set. It contains only degree and child-count features of the two BFS parents q and t. It is the explicit-feature analogue of asking what information becomes available if a one-hop copy brings parent-side counts into the node states.

$$x_{ij}^{(D)} = [x_{ij}^{(A)},\ x_{ij}^{(B\setminus A)},\ x_{ij}^{(C)}],$$

where $x_{ij}^{(B\setminus A)}$ denotes the BFS-tree pair-relation columns added in $x_{ij}^{(B)}$ after the $x_{ij}^{(A)}$ columns. This is (D), all features. It concatenates local features, pair relations from the BFS tree, and BFS-parent degree/child-count features.