From Sequence to Structure: Uncovering Substructure Reasoning in Transformers

Xinnan Dai[1†]*, **Kai Yang**[2†], **Jay Revolinsky**[1], **Kai Guo**[1*],
Aoran Wang[3], **Bohang Zhang**[2], **Jiliang Tang**[1]
[1]Michigan State University, [2]Peking University, [3]University of Luxembourg
{daixinna, revolins, guokai1, tangjili}@msu.edu, yangkai@alumni.pku.edu.cn,
zhangbohang@pku.edu.cn, ralf.wong@outlook.com

Abstract

Recent studies suggest that large language models (LLMs) possess the capability to solve graph reasoning tasks. Notably, even when graph structures are embedded within textual descriptions, LLMs can still effectively answer related questions. This raises a fundamental question: *How can a decoder-only Transformer architecture understand underlying graph structures?* To address this, we start with the substructure extraction task, interpreting the inner mechanisms inside the transformers and analyzing the impact of the input queries. Specifically, through both empirical results and theoretical analysis, we present Induced Substructure Filtration (ISF), a perspective that captures the substructure identification in the multi-layer transformers. We further validate the ISF process in LLMs, revealing consistent internal dynamics across layers. Building on these insights, we explore the broader capabilities of Transformers in handling diverse graph types. Specifically, we introduce the concept of thinking in substructures to efficiently extract complex composite patterns, and demonstrate that decoder-only Transformers can successfully extract substructures from attributed graphs, such as molecular graphs. Together, our findings offer a new insight on how sequence-based Transformers perform the substructure extraction task over graph data.

1 Introduction

It is evident from recent studies that large language models (LLMs) are capable of understanding structured data [31, 22, 15]. For example, when graph structures are presented in textual sequence, LLMs can identify node connections [10, 21], detect graph patterns [3, 7], and compare common subgraphs across a given set [20, 7]. However, transformers, which serve as the backbone of LLMs, are inherently designed for sequential textual data, which does not naturally capture graph structures. This gap raises a fundamental question: How can a sequence-based decoder-only transformer comprehend structured data like graphs?

To answer this question, existing research focuses mainly on basic graph reasoning tasks to build the concept of the mechanism by which transformers understand graph structures [17, 29]. The shortest path is one of the basic tasks [21, 6, 1]. Based on the shortest path task, SLN [5] suggests that a form of spectral navigation implicitly emerges within Transformer layers, enabling global coordination across nodes. Meanwhile, Abulhair et al. [18] and ALPINE [25] argue that Transformers learn to find paths by composing and merging multiple candidate paths based on the provided edge list. However, these studies are limited to linear paths, while real-world graphs often contain more complex, non-linear substructures such as cycles, trees, and other motifs. As a result, existing understandings drawn

*corresponding: {guokai1, daixinna}@msu.edu, † equal contribution, code is available at
`https://github.com/DDigimon/From_Sequence_to_Structure`

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

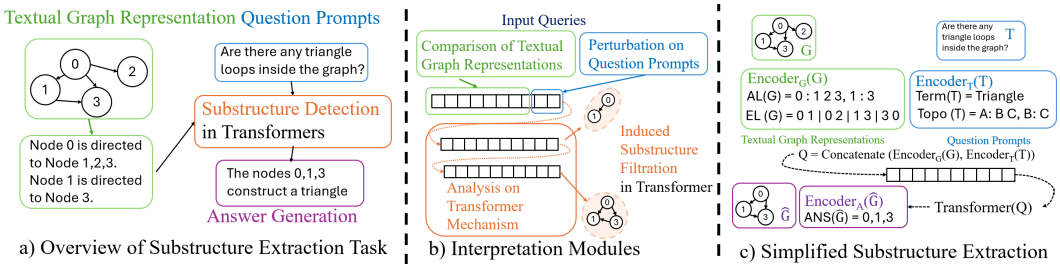


Figure 1: An overview of the interpretation for the substructure extraction task. a) Substructure Extraction Task: The Transformer receives a graph description and a question prompt as input and generates an answer. b) Interpretation Modules: The analysis includes input queries and internal Transformer processing. c) Simplified Substructure Extraction: The extraction process is simplified to highlight the core mechanism.

from path-finding tasks may not generalize to comprehensive graph understanding and are limited to explain why LLMs can do various graph tasks.

In this work, we explore how Transformers tackle the broader challenge of substructure understanding, with a particular emphasis on the task of substructure extraction. Building on the use of LLMs for substructure extraction [7, 3], Figure 1a) illustrates the overall process with Transformers: Transformer-based models receive a query prompt, which is composed of a textual graph representation and a question prompt as input. Then, they identify the relevant substructure, and generates an answer for the given graph.

To investigate how Transformers derive answers from input tokens, in Section 3 we conduct empirical and theoretical analyses of train-from-scratch Transformers, focusing on the internal Transformer mechanisms and input queries, as shown in Figure 1b). To understand the internal behavior of the model, we introduce a new perspective, Induced Substructure Filtration (ISF), which suggests that Transformers perform a layer-wise node aggregation process to detect substructures.

To verify the reliability of our interpretation modules, we demonstrate that our approach also applies to understanding LLM behavior and argue that it can inform the development of future methods. In Section 4, we show that our explanation for Transformers aligns with the behaviors observed in LLMs, particularly in how they tackle various textual graph representations and perform graph extraction tasks. Furthermore, in Section 5, we explore the potential of Transformers in graph understanding, building on insights from our interpretation modules. We argue that it is reasonable to extend Transformers to handle attributed graphs, such as molecular graphs. Finally, we introduce the Thinking-In-Substructure framework, which enhances Transformers' capabilities in complex graph reasoning tasks. In summary, we offer a new perspective on how Transformers understand graph structures from sequential inputs. Our key contributions are summarized as follows:

1. We provide insights into how Transformers extract substructures, based on experiments and theory, focusing on internal mechanisms and input queries.
2. We propose Induced Substructure Filtration (ISF) to explain how Transformers identify substructures across layers.
3. We show that our interpretation is applicable to LLMs, explaining their behaviors in graph tasks and supporting the extension of Transformers to attributed graphs and more complex graph reasoning.

2 Preliminary

We combine theoretical insights and experimental results to demonstrate how Transformers solve the substructure extraction task. To support this, we briefly introduce the core notations and definitions used in this work and provide an overview of the experimental setup for the following empirical studies.

2.1 Problem formulations

Substructure extraction in transformers Although we prompt LLMs to interpret natural language sentences to answer substructure-related questions, we simplify the process by converting these sentences into symbolic tokens. This enables us to analyze and train Transformers from scratch. The simplified process is illustrated in Figure 1 c). Given a graph $G = \{V, E\}$ and a question prompt T, we encode them using $\mathrm{Encoder_G}$ and $\mathrm{Encoder_T}$ respectively to obtain simplified sentence sequences. These are then concatenated, with the encoded question placed after the graph representation, forming the input query Q. The objective is to extract the set of isomorphic subgraphs $\hat{G} = \{g_1, g_2, \cdots, g_s\}$, where each g_i represents an instance of the desired substructure within the input graph. The overall process is defined as: $\mathrm{Encoder_A}(\hat{G}) = \mathrm{Transformers}(Q) = \mathrm{Transformers}(\mathrm{Encoder_G}(G), \mathrm{Encoder_T}(T))$. We introduce each encoder of this framework in the following paragraphs.

Textual graph representations ($\mathrm{Encoder_G}$) To input graphs into a Transformer, prior work [10, 7] often converts them into textual sequences using either the Adjacency List (AL) or Edge List (EL). Specifically, for a graph $G = \{V, E\}$ and a vertex $v_i \in V$, the AL format captures its neighborhood $N(v_i) = \{v \in V \mid (v_i, v) \in E\} = \{v_i^1, \cdots, v_i^{m_i}\}$, where m_i is the number of neighbors of nodes v_i. In the textual representation, each node and its neighbors are formatted as a sentence: the central node and its neighbors are separated by a colon ":", and different such groups are separated by commas ",", which formulated as:

$$\mathsf{AL}(G) = (v_1; ":"; v_1^1; \cdots; v_1^{m_1}; ","; \cdots; ","; v_n; ":"; v_n^1; \cdots; v_n^{m_n}).$$

Instead of focusing on central nodes, the EL format enumerates all possible edges $(v_i, v_j) \in E$. Each edge pair is separated by a vertical bar "|". The representation of EL is formulated as:

$$\mathsf{EL}(G) = (v_1; v_1^1; "|"; \cdots; "|"; v_1; v_1^{m_1}; "|"; \cdots; "|"; v_n; v_n^1; "|"; \cdots; "|"; v_n; v_n^{m_n}).$$

The details of the definitions are in the Definition D.1 and Definition D.2 in Appendix D.1.

Question prompt ($\mathrm{Encoder_T}$) Next, we define the question prompt to determine which substructures should be extracted from the input graph. This prompt, denoted as instruction T, can be either terminology-based or topology-based, as described in [7]. If the substructures are well-known, such as a "triangle", they can be defined using either terminology or topological instructions, represented as $\mathsf{Term}(T) = (\text{triangle})$ and $\mathsf{Topo}(T) = (\mathrm{A} : \mathrm{BC}, \mathrm{B} : \mathrm{C})$, respectively. However, in most cases, the substructures are not clearly defined by terminology, so we rely on topology-based definitions.

Answer generation ($\mathrm{Encoder_A}$) The output of the Transformer is a text sequence. However, this sequence must correspond to a unique substructure. To align the substructures with the text output, we constrain the Transformer to output the node sets for each substructure, separated by commas. Formally, the output is represented as: $\mathsf{ANS}(\hat{G}) = (v_1^{g_1}, v_2^{g_1}, \ldots, v_w^{g_1}, ",", \ldots, ",", v_1^{g_s}, v_2^{g_s}, \ldots, v_w^{g_s})$, where each group $\{v_1^{g_i}, \ldots, v_w^{g_i}\}$ denotes the nodes in subgraph g_i, and commas "," are used to delimit different substructures.

2.2 Experiment settings

Transformer training We train Transformer models using the same architecture as GPT-2 but in a lightweight version, with only 384 hidden dimensions and a small number of layers depending on the tasks. The details for each task are shown in the Appendix E. During training, the model is optimized only to predict $\mathsf{ANS}(\hat{G})$. For evaluation, we use accuracy as the metric. A predicted answer is considered correct only if the $\mathsf{ANS}(\hat{G})$ is exactly the same with the ground truth.

Dataset setting We generate over 5 million directed graphs, with node counts ranging from 4 to 16 and edge counts from 3 to 120. The graphs are constructed based on specific requirements detailed in the following empirical studies. To prevent result copying from the same graphs, we ensure that the graphs in the training and testing sets are non-isomorphic.

Table 1: Transformers extract the substructures from the given graph sequence

# Training	# Layer	Triangle	Path	Square	Diagonal	T_triangle	F_Triangle	Diamond	Pentagon	House
										
100K	2	0.5301 ± 0.06	0.5534 ± 0.03	0.1936 ± 0.04	0.1163 ± 0.00	0.1911 ± 0.03	0.2877 ± 0.03	0.0656 ± 0.01	0.3628 ± 0.01	0.3705 ± 0.00
	3	0.9662 ± 0.00	0.8066 ± 0.02	0.3991 ± 0.00	0.4417 ± 0.07	0.4974 ± 0.00	0.5329 ± 0.04	0.1635 ± 0.03	0.5638 ± 0.01	0.5603 ± 0.00
300K	3	0.9948 ± 0.00	0.9195 ± 0.01	0.7247 ± 0.01	0.6313 ± 0.07	0.7775 ± 0.02	0.7831 ± 0.06	0.6189 ± 0.02	0.7063 ± 0.05	0.7455 ± 0.03
	4	0.9947 ± 0.00	0.9493 ± 0.02	0.9403 ± 0.02	0.9140 ± 0.02	0.9080 ± 0.03	0.8097 ± 0.02	0.8765 ± 0.02	0.8634 ± 0.00	0.8386 ± 0.04
400K	4	0.9802 ± 0.02	0.9802 ± 0.01	0.9620 ± 0.00	0.9596 ± 0.00	0.9287 ± 0.02	0.8534 ± 0.01	0.9048 ± 0.03	0.8612 ± 0.01	0.8023 ± 0.05
	5	0.9977 ± 0.00	0.9948 ± 0.00	0.9679 ± 0.02	0.9707 ± 0.01	0.9430 ± 0.04	0.8750 ± 0.02	0.9306 ± 0.02	0.8922 ± 0.01	0.8530 ± 0.02

3 Interpretations for Substructure Extraction in Transformers

In this section, we present our insights into how Transformers perform substructure understanding. We focus on two main aspects: the internal mechanisms of Transformers in solving the substructure extraction task, discussed in Section 3.1, and the impact of input query formulation on extraction performance Section 3.2.

3.1 Induced Substructure Filtration in Transformer

In this subsection, we introduce how Transformers solve the substructure extraction task. First, we show that Transformers can extract substructures of diverse shapes, as detailed in Section 3.1.1. We then analyze the underlying mechanism and propose the ISF process in Section 3.1.2. Finally, we demonstrate how ISF generalizes to cases with multiple substructures of varying numbers and shapes in Section 3.1.3.

3.1.1 Single substructure extraction

We begin with the Single-Shape-Single-Num case, evaluating whether Transformers can extract a specific target substructure from a given graph whose scale and shape may vary. The selected substructures contain 3 to 5 nodes and 3 to 6 edges. We also investigate the effects of dataset size and the number of Transformer layers. To this end, we vary the training set size from 100K to 400K and the number of Transformer layers from 2 to 5. For each substructure, we evaluate the extraction accuracy on 30K test graphs, averaging results over three runs. Table 1 shows the results for various substructure extraction tasks.

The Transformers are capable to extract the target shape of substructures from the given graph with at least 2 layer transformers, achieving over 85% accuracy. However, different substructures exhibit varying requirements in terms of both data scale and model depth. For instance, the 3-cycle (triangle) structure can achieve 99% accuracy with just 3 layers and 100K training examples, while the 5-cycle (pentagon) structure requires over 5 layers and at least 400K examples to attain comparable performance. Furthermore, we observe that the minimum number of Transformer layers required to achieve 85% accuracy correlates with the number of nodes in the substructure. For example, all 4-node substructures can achieve promised results with 4-layer transformers, while 5-node substructures need 5-layer transformers. This suggests that the number of layers is a crucial factor in a Transformer's ability to understand graph structures. To understand how the number of layers affects a Transformer's grasp of graph structures, we analyze substructure extraction across layers in the following subsections.

3.1.2 Induced Subgraph Filtration

Visualization Results We visualize token embeddings to better understand how Transformers extract substructures. Since decoder-only Transformers process input left to right [2], we use the final token embeddings to reflect their graph understanding. We apply t-SNE to project these embeddings into a 2D space, labeling each graph by its substructure answer, which is represented by the node IDs as illustrated in $\mathsf{ANS}(\hat{G})$. As an example, we use the square substructure extraction task, shown in Figure 2. The legends are the node IDs. For example, "0431" indicates that the model first identifies node 0 (with out-degree 2), followed by its neighbor 4 (out-degree 1), then node 3 (a neighbor of 4), and finally node 1 (a neighbor of both 3 and 0)

The visualization results reveal how Transformers identify substructures across layers, with graphs sharing similar answers gradually clustering together. Although the visualization targets the final graph token, the substructure answers are already determined by the last layer before the generation

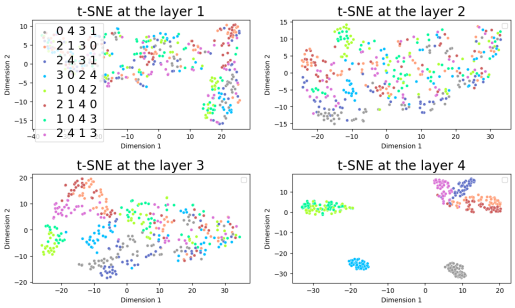


Figure 2: Visualization across 4 layers. We show the node ID distributions of target substructures. Legends indicate the node IDs

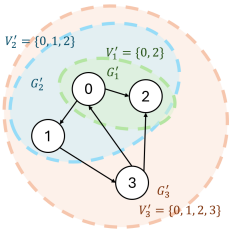


Figure 3: 4-Node 3-Filtration and Induced Subgraph Filtration

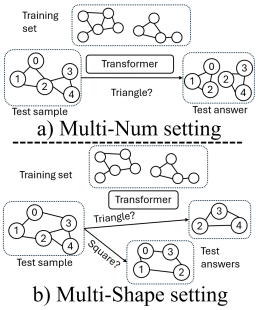


Figure 4: Tasks in Simultaneous detection

step.We also observe that substructures form progressively across layers. For example, in layer 2, graphs with substructure ID '2431' (dark blue) begin clustering near '0431' (grey) and '2413' (purple), sharing '31' and '24'. By layer 3, '2431' moves closer to '0431', which shares '431'. In the final layer, substructure types are clearly separated. Since outputs follow a left-to-right order, those with similar starting tokens cluster together. Transformers infer substructures before generation, progressively organizing substructures across layers.

Theoretical modeling We further provide a theoretical framework for this process by introducing filtrations to formalize progressive substructure extraction. Using the square extraction visualization as an example, we model it as a 4-node, 3-filtration process, indicating that the target substructure contains 4 nodes and requires 3 filtration steps, each corresponding to an induced subgraph. More generally, we define this framework as a k-Node m-Filtration, referred to as Induced Subgraph Filtration, as defined in Definition 3.1.

Definition 3.1 (k-Node m-Filtration and Induced Subgraph Filtration)**.** A *k-node m-filtration* on V' ($|V'| = k$) is $\mathcal{F}(V') = (V_1', \ldots, V_m')$ where $\varnothing \neq V_1' \subseteq \cdots \subseteq V_m' = V'$. For $G' = (V', E')$, this yields an *induced subgraph filtration* $(G_1', \ldots, G_m')$ where $G_i' = G'[V_i']$.

Figure 3 illustrates the concept defined in Definition 3.1. We model the extracted substructure as $G' = (V', E')$ with $|V'| = k$, and represent the extraction process using an m-Filtration, where m denotes the number of gathering operations required for the Transformers to identify the substructure.

Further, to capture the matches of G' in $G = (V, E)$, we define a Substructure Isomorphism Indicator Tensor.

Definition 3.2 (Subgraph Isomorphism Indicator Tensor)**.** For graphs $G = (V, E)$ ($|V| = n$) and $G' = (V', E')$ ($|V'| = k$), the *subgraph isomorphism indicator tensor* $\mathcal{T}(G, G')$ is k-dimensional ($n \times \cdots \times n$) where its entry for an ordered k-tuple of vertices $(v_{j_1}, \ldots, v_{j_k})$ from V is 1 if these vertices induce a subgraph isomorphic to G' (via a predefined mapping $v_p' \mapsto v_{j_p}$ for $p = 1, \ldots, k$), and $\mathcal{T}_{j_1,\ldots,j_k} \leq 0$ otherwise (see Definition D.5 for details).

Theorem 3.3 (proof in Appendix D.3) shows that Transformers can progressively compute $\mathcal{T}(G, G')$ for each substructure along the filtration. $O(n^k)$ is the hidden dimension needed for the Transformer to check all k-node subgraphs in an n-node graph.

Theorem 3.3 (Expressiveness for Progressive Identification)**.** *Given a k-node m-filtration $\mathcal{F}(V')$ on $V' = \{v_1', \ldots, v_k'\}$. For any directed graphs $G = (V, E)$ ($|V| = n$) and $G' = (V', E')$, a log-precision Transformer with $m + 2$ layers, constant heads, and $O(n^k)$ hidden dimension can output* $\mathsf{vec}(\mathcal{T}(G, G'[V_i']))$ *at layer $i + 2$ for $i \in \{1, \ldots, m\}$.*

Furthermore, the Transformer extracts the unique instance of G', meaning the answer is uniquely determined by the given graph representation and question prompt, and $\mathcal{T}(G, G')$ contains exactly one entry equal to 1. This leads to Assumption 3.4 and Theorem 3.5. With this condition, the substructure extraction task is solvable for transformers.

Assumption 3.4 (Single-Shape-Single-Num)**.** For graphs G, G', there is a *unique* k-tuple of indices $(i_1, \ldots, i_k)$ for which $\mathcal{T}(G, G')_{i_1,\ldots,i_k} = 1$.

Theorem 3.5 (Expressiveness for Pattern Extraction)**.** *Under Assumption 3.4, for directed graphs $G = (V, E)$ ($|V| = n$) and $G' = (V', E')$ ($|V'| = k$), a log-precision Transformer with constant depth, constant heads, and $O(n^k)$ hidden dimension can output the unique k-tuple of vertices $(v_{i_1}, \ldots, v_{i_k})$ for which $\mathcal{T}(G, G')_{i_1,\ldots,i_k} = 1$.*

Remark 3.6. Theorems 3.3 and 3.5 hold for various input graph representations (e.g., adjacency lists $\mathsf{AL}(G)$ or edge lists $\mathsf{EL}(G)$), as formally defined in Definitions D.1 and D.2.

3.1.3 Simultaneous detection of multiple substructures

As graphs often contain multiple and diverse substructures, we further demonstrate how the ISF process adapts to such scenarios. Specifically, we evaluate whether a 4-layer Transformer can accurately detect both repeated and differently shaped substructures. To this end, we design the Single-Shape-Multi-Num and Multi-Shape-Single-Num evaluation settings, with the pipeline illustrated in Figure 4.

Single-Shape-Multi-Num As shown in Figure 4 a), the Single-Shape-Multi-Num task involves training and testing on graph sets where each sample may contain multiple target substructures—up to five in total. This task evaluates whether Transformers can successfully extract all target substructures within a single graph. We define the task in Definition 3.7.

Definition 3.7 (Single-Shape-Multi-Num Extraction)**.** The Single-Shape-Multi-Num extraction task requires a model to output all k-tuples of vertices $(v_{i_1}, \ldots, v_{i_k})$ corresponding to occurrences of directed graph $G' = (V', E')$ ($|V'| = k$) in $G = (V, E)$ ($|V| = n$). Formally, the objective is to output all tuples satisfying $\mathcal{T}(G, G')_{i_1,\ldots,i_k} = 1$, where $\mathcal{T}(G, G')$ is Subgraph Isomorphism Indicator Tensor defined in Definition 3.2.

As shown in Figure 5, Both triangles and square detections can achieve over 85% accuracy. The number of substructures has little impact on the Transformers' ability. They can identify multiple patterns at once. We further analyze examples with two substructures and find that answers are still often determined before the final generation step (Figure 7). The theoretical explanation is provided in Theorem 3.8.

Theorem 3.8 (Expressiveness for Single-Shape-Multi-Num Extraction)**.** *Fix integers $n \geq k \geq 1$. There exists a log-precision Transformer with constant depth, constant number of attention heads, and $O(n^k)$ hidden dimension that can complete Single-Shape-Multi-Num Extraction defined in Definition 3.7 for directed graphs $G = (V, E)$ ($|V| = n$) and $G' = (V', E')$ ($|V'| = k$).*

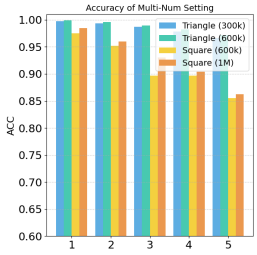


Figure 5: The Multi-Num setting results

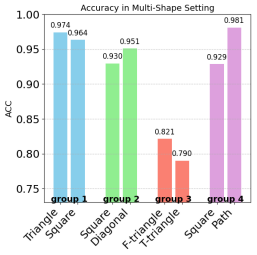


Figure 6: The Multi-Shape setting results

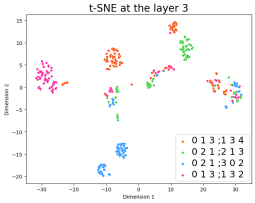


Figure 7: Transformers has organized the answers before the generate the answers.

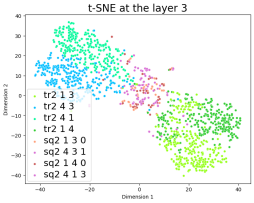


Figure 8: Triangles (tr) are identified at layer 3 when trained with squares (sq).

Multi-Shape-Single-Num As illustrated in Figure 4 b), the Multi-Shape-Single-Num task involves training and testing on graph sets where each sample may contain multiple substructures of different shapes. This task evaluates whether Transformers can identify diverse substructure types within a single graph defined in Definition 3.9.

Definition 3.9 (Multi-Shape-Single-Num Extraction)**.** The Multi-Shape-Single-Num extraction task requires a model to find all the occurrences for any directed graph $G' = (V', E')$ ($|V'| \leq k$) in $G = (V, E)$ ($|V| = n$) satisfying Assumption 3.4. Formally, the objective is to output the *unique* k'-tuple of vertices $(v_{i_1}, \ldots, v_{i_{k'}})$ for which $\mathcal{T}(G, G')_{i_1,\ldots,i_{k'}} = 1$, where $\mathcal{T}(G, G')$ is Subgraph Isomorphism Indicator Tensor defined in Definition 3.2.

We create this task by mixing single-shape training data and dividing it into four groups (details in Appendix E.2), as shown in Figure 6. Each substructure can be extracted independently, guided by its specific question prompt. Therefore, in visualization, we take the embedding of the last input query token rather than the last graph representation token. From Figure 8, we find that Transformers often identify simpler substructures, like triangles, in layer 3, instead of delaying all predictions to the final layer (the Transformer has 4 layers in this setting). We summarize the mechanism in Theorem 3.10.

Theorem 3.10 (Expressiveness for Multi-Shape-Single-Num Extraction)**.** *Fix integers $n \geq k \geq 1$. There exists a log-precision Transformer with constant depth, constant heads, and $O(n^k)$ hidden dimension that can complete Multi-Shape-Single-Num Extraction defined in Definition 3.9 for a directed graph $G = (V, E)$ ($|V| = n$) and any target subgraph $G' = (V', E')$ with $|V'| = k' \leq k$ satisfying Assumption 3.4.*

These two properties form the foundation for understanding how decoder-only Transformers decompose complex graphs into simpler ones for substructure extraction, as discussed in Section 5.1.

3.2 Impact of Input Query Formulation

As illustrated in Section 2, the input query consists of two components: a text-based graph representation and a question prompt. In Section 3.2.1 and Section 3.2.2, we discuss how Transformers perform the substructure extraction task based on these inputs.

3.2.1 Text-Based Graph Representations

We start with the comparison of different text-based graph representation methods. Specifically, we focus on the two basic methods, which are the neighborhood-based AL and the edge-based EL. To ensure controllable input lengths, we conduct a toy experiment using graphs with 4 to 8 nodes, keeping both representations within a 100-token limit. We then vary the number of Transformer layers to extract substructures (e.g., triangle, square, or pentagon) from the graph representations. Experimental details are provided in the Table 7 in Appendix E.1 and the results are shown in Figure 9.

The experimental results indicate that both AL and EL formats allow Transformers to extract substructures from text-based graph representations. As the size of the target substructure increases, both formats require more Transformer layers to achieve more than 80% accuracy. As mentioned in Remark 3.6, the theoretical results hold for both AL and EL formats. The intuition is that both formats can be transformed into the same binary adjacency matrix $A(G)$, which encodes the structure of graph G. This matrix is then vectorized as $\mathsf{vec}(A(G))$ for processing within the model, where we provides the details in Lemma D.6 in Appendix D.2.

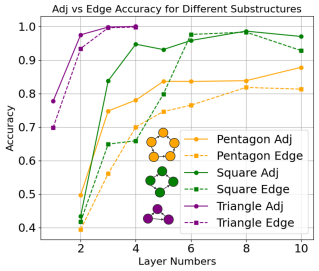


Figure 9: Using AL and EL to predict substructures with varying numbers of Transformer layers.

Although we theoretically show that EL and AL are equivalent in representational power, experimental results reveal that EL requires more Transformer layers to achieve comparable performance. This may be because EL inherently requires more tokens to explicitly represent all edges, whereas AL encodes the same information with fewer tokens and additional padding, benefiting from its more compact structure. In practice, for a fully connected graph, the AL needs $2 \times |V|^2 - |V|$ tokens, while the EL needs $3 \times (|V|^2 - |V|)$ tokens. Therefore, for efficiency, we mainly adopt AL in our discussions. Appendix E.3 provides the details of the training efficiency comparison on AL and EL.

3.2.2 Question prompt encoders

We explore how question prompts influence structural understanding in substructure extraction. The question prompt T claims the target substructure, expressed as either terminology-based $\mathsf{Term}(T)$ (e.g., "triangle") or topology-based $\mathsf{Topo}(T)$. In $\mathsf{Topo}(T)$, we use AL-style descriptions with symbolic node labels, e.g., a triangle as $\mathsf{Topo}(T) = (A : BC, B : C)$.

We construct a mixed data set to train the Transformer, following the setup in Section 3.1.3, but balance it with equal samples using topology-based and terminology-based prompts, denoted "Term", "Topo1" and "Topo2" in Table 2. The "Term" uses semantic labels (e.g., "Triangle"), while "Topo1"

Table 2: Results on different question prompts. “p" means the pad token.

Mixture training	Term	ACC	Topo1	ACC	Topo2	ACC	Symbol-level	ACC	Token-level	ACC
Group1	Triangle	0.9782	A:BC,B:C	0.9794	B:AC,A:C	0.9166	: , :	0.9152	C/D	0.7074 / 0.1027
	Square	0.8478	A:D,C:BA,D:B	0.8494	B:AD,A:C,C:D	0.8500	: , : , :	0.7444	C/D	0.7532 / 0.8470
Group2	Diagonal	0.9082	A:BCD,C:D,D:B	0.2332	B:D,C:ABD,D:A	0.7354	p:pp,p:p,p:p	0.7086	A/C	0.8566 / 0.2991
	Square	0.8810	A:BC,C:D,D:B	0.8691	B:AD,A:C,C:D	0.9037	p:p,p:ppp,p:p	0.7106	A/C	0.1271 / 0.9094

gives direct node connections and "Topo2" describes the same structure with shuffled node names. To examine how Transformers align different descriptions with graph inputs, we introduce two types of perturbations: symbol-level and token-level, where these tokens are used directly as question prompts. Symbol-level perturbations test the impact of structural phrasing, while token-level perturbations assess reliance on specific tokens within topology prompts. Results are reported in Table 2. In token-level perturbation, we use two different tokens to examine whether they have distinct impacts on Transformer performance.

The results show that Transformers can use both terminology- and topology-based prompts, achieving over 70% accuracy in each case. However, terminology-based prompts perform better, reaching over 85% in Group 2. For example, Topology-based questions show limitations, struggling to represent diagonal substructures accurately. Perturbation results suggest that predictions often rely on specific symbolic cues or tokens rather than full structural understanding. For instance, in Group 1, triangles are identified via symbolic patterns, while in Group 2, diagonal and square structures are distinguished by tokens like “A” and “C.”. This suggests that Transformers do not explicitly learn full structural representations or map them back to the original graphs for answers. Instead, they abstract substructure concepts using a series of key tokens.

4 Consistency in LLM Graph Understanding Behavior

As discussed in Section 3, we identify three key findings: In terms of the Transformer mechanism, (1) Transformers perform the ISF process to extract substructures simultaneously. Regarding input formulation, (2) both EL and AL can represent the adjacency matrix $A(G)$, though EL may perform slightly worse than AL due to sequence length limitations, and (3) Transformers tend to abstract substructures into a sequence of tokens in the question prompt, rather than fully capturing the underlying topological concepts. Since decoder-only Transformers are a common architecture in modern LLMs, we further evaluate whether LLMs exhibit the ISF process during substructure extraction, and whether our understanding of input formulation can explain their behavior.

Induced Subgraph Filtration To investigate whether LLMs exhibit the ISF process, we visualize the fine-tuned LLaMA 3.1-8B-Instruct model on a triangle detection task (details in Appendix E.4.2), as shown in Figure 10. The results align with our findings in Section 3.1: the model often identifies the correct answer before generating it, and deeper layers better distinguish between similar answers. For example, the responses of the substructures in node ID ‘302’ and ‘304’ become more separable at layer 23 than at layer 17. We quantify this trend in Figure 11, where Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) scores increase with depth. However, unlike Transformers trained from scratch, which only predict answers, LLMs can generate explanatory content. For instance, 23% of responses include Python code, while others describe node relationships in the graph. As a result, ARI and NMI scores slightly decrease in the ending layers.

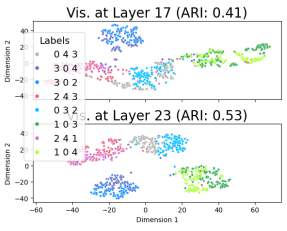


Figure 10: Visualization on Llama3.1-8B-Instruct

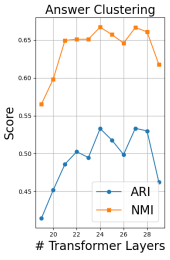


Figure 11: ARI and NMI across the layers.

Text-based Graph Representation Current evaluations have discussed the effect of how different graph representations influence the LLMs in graph reasoning tasks. For example, GraphMem [31] demonstrates that LLMs possess the ability to transfer knowledge across different graph descriptions. This is possible because both AL and EL can be mapped to $\mathsf{vec}(A(G))$, making them share the representations at the graph representation level. However, since the EL typically requires more tokens to describe a graph and LLMs have limitations in handling long contexts, prior studies [10, 6] report that EL representations sometimes perform worse than AL descriptions.

Question Prompt In the context of substructure understanding, GraphPatt [7] introduces both terminology-based and topology-based descriptions for the substructure extraction task, showing that terminology-based prompts generally lead to better performance. Moreover, a single terminology concept can correspond to multiple diverse topology-based descriptions reported in [7].

5 Understanding on Complex Graphs

In Section 3, we highlight the ISF process as a key mechanism enabling Transformers to solve the substructure extraction task. Moreover, we observe that when text-based descriptions can be mapped to the adjacency matrix $A(G)$, Transformers can perform substructure extraction on the given graph. Building on these insights, we explore broader applications: in Section 5.1, we introduce a new method for efficiently reasoning over composite substructures and in Section 5.2, we examine how Transformers adapt to attributed graphs.

5.1 Thinking in substructures

As introduced in Section 3, Transformers apply the ISF process to extract substructures. This process runs synchronously across different patterns, with smaller ones being easier to detect. Besides, complex structures often consist of simpler components, which we refer to as decomposing substructures. Building on these observations, we propose the Thinking-in-Substructure (Tins) method to explain how decoder-only Transformers solve complex substructure reasoning tasks. For example, [7] reports that reasoning language models decompose a house pattern into a triangle and a square to search for the substructures. We reformulate the answer generation part as $\mathsf{ANS}_{\mathsf{Tins}}(\hat{G}) = (\{P_1\}, \{P_2\}, \ldots, \{P_t\}, < \text{ANS} >, \mathsf{ANS}(\hat{G}))$, where $\{P_i\}$ is the collection of each decomposing structure and $< \text{ANS} >$ is a special token to indicate that the followings are final answers. This decomposition reduces the extraction complexity from $O(n^k)$ to $O(n^q)$, where q and k are the maximum size of decomposing substructures and target substructures, respectively, with $q < k$. We show the proof in Theorem D.13 in Appendix D.4.

To verify the efficiency of Tins, in the experiment, we design 4 different composite substructures with their decomposition process trained with 100K samples. The experiment settings are in Appendix E.4.3. The results suggest that Tins can help transformers significantly improve the performance with limited training data. The overall performance can increase 10%, and in the 3 layer for diagonal structure, the performance increases about 46% percent.

Table 3: The results of Thinking-in-substructures (Tins)

Substructures	Directly Preds 4 layer	Directly Preds 3 layer	Decomposition	Tins 4 layer	Tins 3 layer
Diagonal	0.6314	0.1998		0.8648	0.6606
Diamond	0.4756	0.1288		0.7792	0.4338
House	0.5887	0.5640		0.8066	0.6678
Complex	0.1182	0.1208		0.2268	0.2124

Table 4: molecular graphs

Functional group		# Node	ACC
C-O(H)		9	0.9207
COO(H)		121	0.9159
$C_6(H_6)$		121	0.7245
Mix		121	0.8946

5.2 Attributed graphs

As shown in Theorem 3.5, the only condition required for Transformers to extract substructures is $\mathcal{T}(G, G') = 1$. Therefore, if node features are uniquely assigned to ensure a distinct graph representation for each graph, Transformers can extract substructures while incorporating these features, as discussed in Theorem D.16. Taking the AL-based feature description as an example, we define the attributed graph representation as $\mathsf{AL}_{\mathsf{f}}(G) = (v_1 f_1; \text{“:”}; v_1^1 f_1^1; \cdots; v_1^{m_1} f_1^{m_1}; \text{“,”}; \cdots; \text{“,”}; v_n f_n; \text{“:”}; v_n^1 f_n^1; \cdots; v_n^{m_n} f_n^{m_n})$. , where f_i is the node features.

We use molecular graphs as examples to test how well Transformers understand attributed graphs with the attributed AL list $\mathsf{AL}_{\mathsf{f}}(G)$, where we set node feature f_i as atoms. In the experiments, the transformers predict the positions of functional groups like Hydroxyl (C-O(H)), Carboxyl (COO(H)),

and Benzene Ring $C_6(H_6)$. We also use a mixed training setup where Hydroxyl and Carboxyl are combined as "Mix". Molecules contain 1 to 4 target groups. Details are in Appendix E.4.4 Results (see Table 4) show that Transformers perform well, even with mixed training, aligning with our discussion in Theorem 3.8 and Theorem 3.10.

6 Related work

Evaluations for LLMs in graph understanding Benchmarks reveal that LLMs can recover graph structure from text. NLGraph showed basic reachability and shortest-path competence [21]; InstructGraph and GraphArena scaled tasks and graphs, with graph-aware verbalization and instruction-tuning boosting accuracy even on million-node inputs [22, 20]. GPT-4 few-shot can rival GNNs on node classification but is sensitive to token order [26]. Parallel work builds graph foundation models: GraphToken injects learned tokens, yielding substantial performance gains [32]; Graph2Token aligns molecules with text; and recent surveys chart cross-domain transfer [27, 23]. These two threads, task benchmarks and graph-biased LLMs, form the empirical backdrop for our ISF theory.

Understanding transformers for graphs The mechanisms originate from graph learning methods, with detailed comparisons of graph neural networks, graph transformers, and decoder-only transformers in Appendix C. Theory now probes how vanilla Transformers perform graph reasoning. Log-depth models suffice, and are necessary, for connectivity and cycles [17], while width can trade for depth [29]. ALPINE [24] shows a GPT layer embeds adjacency and reachability, validating on planning tasks; two-layer decoders trained on shortest-path learn spectral line-graph embeddings instead of Dijkstra-style rules [5]. Surveys relate attention power to Weisfeiler–Lehman bounds and over-squashing limits [13, 19]; scalable variants such as AnchorGT mitigate $O(n^2)$ cost without losing accuracy [34]. Hierarchical distances or sparse global attention keep Transformers competitive on large or molecular graphs [9, 8, 30]. Collectively, these studies view attention heads as induced-neighbourhood selectors—exactly the mechanism ISF formalises via filtration depth.

7 Conclusion

This paper explores how decoder-only Transformers perform substructure reasoning over graphs represented as text. We propose ISF to model how substructures are progressively identified across layers. Our analysis shows that extraction accuracy depends on substructure size, model depth, and input format. We further validate ISF in LLMs, revealing consistent internal mechanisms. Extending this framework, we introduce the Tins method to handle composite and attributed graphs. These findings provide a unified view of how Transformers and LLMs reason over structured data.

Acknowledgement

Xinnan Dai, Jay Revolinsky, Kai Guo, and Jiliang Tang are supported by the National Science Foundation (NSF) under grant numbers CNS2321416, IIS2212032, IIS2212144, IIS 2504089, DUE2234015, CNS2246050, DRL2405483 and IOS2035472, the Michigan Department of Agriculture and Rural Development, US Dept of Commerce, Gates Foundation, Amazon Faculty Award, Meta, NVIDIA, Microsoft and SNAP.

References

[1] Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In *International Conference on Machine Learning*, pages 2296–2318. PMLR, 2024.

[2] Lijie Chen, Binghui Peng, and Hongxun Wu. Theoretical limitations of multi-layer transformer. *arXiv preprint arXiv:2412.02975*, 2024.

[3] Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. Graphwiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 353–364, 2024.

[4] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020.

[5] Andrew Cohen, Andrey Gromov, Kaiyu Yang, and Yuandong Tian. Spectral journey: How transformers predict the shortest path. *arXiv preprint arXiv:2502.08794*, 2025.

[6] Xinnan Dai, Qihao Wen, Yifei Shen, Hongzhi Wen, Dongsheng Li, Jiliang Tang, and Caihua Shan. Revisiting the graph reasoning ability of large language models: Case studies in translation, connectivity and shortest path, 2024. URL `https://arxiv.org/abs/2408.09529`.

[7] Xinnan Dai, Haohao Qu, Yifei Shen, Bohang Zhang, Qihao Wen, Wenqi Fan, Dongsheng Li, Jiliang Tang, and Caihua Shan. How do large language models understand graph patterns? a benchmark for graph pattern comprehension. In *The Thirteenth International Conference on Learning Representations*, 2025.

[8] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[9] Vijay Prakash Dwivedi, Yozen Liu, Anh Tuan Luu, Xavier Bresson, Neil Shah, and Tong Zhao. Graph transformers for large graphs. *arXiv preprint arXiv:2312.11109*, 2023.

[10] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[11] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=qHrADgAdYu`.

[12] Guhao Feng, Kai Yang, Yuntian Gu, Xinyue Ai, Shengjie Luo, Jiacheng Sun, Di He, Zhenguo Li, and Liwei Wang. How numerical precision affects mathematical reasoning capabilities of llms, 2024. URL `https://arxiv.org/abs/2410.13857`.

[13] Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers. *arXiv preprint arXiv:2302.04181*, 2023.

[14] Luis Müller, Daniel Kusuma, Blai Bonet, and Christopher Morris. Towards principled graph transformers. *Advances in Neural Information Processing Systems*, 37:126767–126801, 2024.

[15] Miao Peng, Nuo Chen, Zongrui Suo, and Jia Li. Rewarding graph reasoning process makes llms more generalized reasoners. *arXiv preprint arXiv:2503.00845*, 2025.

[16] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[17] Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. Understanding transformer reasoning capabilities via graph algorithms. *Advances in Neural Information Processing Systems*, 37:78320–78370, 2024.

[18] Abulhair Saparov, Srushti Pawar, Shreyas Pimpalgaonkar, Nitish Joshi, Richard Yuanzhe Pang, Vishakh Padmakumar, Seyed Mehran Kazemi, Najoung Kim, and He He. Transformers struggle to learn to search. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=9cQB1Hwrtw`.

[19] Ahsan Shehzad, Feng Xia, Shagufta Abid, Ciyuan Peng, Shuo Yu, Dongyu Zhang, and Karin Verspoor. Graph transformers: A survey. *arXiv preprint arXiv:2407.09777*, 2024.

[20] Jianheng Tang, Qifan Zhang, Yuhan Li, and Jia Li. Grapharena: Benchmarking large language models on graph computational problems. *arXiv preprint arXiv:2407.00379*, 2024.

[21] Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36:30840–30861, 2023.

[22] Jianing Wang, Junda Wu, Yupeng Hou, Yao Liu, Ming Gao, and Julian McAuley. Instructgraph: Boosting large language models via graph-centric instruction tuning and preference alignment. *arXiv preprint arXiv:2402.08785*, 2024.

[23] Runze Wang, Mingqi Yang, and Yanming Shen. Bridging molecular graphs and large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(20):21234–21242, 2025.

[24] 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. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 119662–119688. Curran Associates, Inc., 2024. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/d848cb2c84f0bba7f1f73cf232734c40-Paper-Conference.pdf`.

[25] 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. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[26] Yuxiang Wang, Xinnan Dai, Wenqi Fan, and Yao Ma. Exploring graph tasks with pure llms: A comprehensive benchmark and investigation. *arXiv preprint arXiv:2502.18771*, 2025.

[27] Yuxiang Wang, Wenqi Fan, Suhang Wang, and Yao Ma. Towards graph foundation models: A transferability perspective. *arXiv preprint arXiv:2503.09363*, 2025.

[28] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[29] Gilad Yehudai, Clayton Sanford, Maya Bechler-Speicher, Orr Fischer, Ran Gilad-Bachrach, and Amir Globerson. Depth-width tradeoffs in algorithmic reasoning of graph tasks with transformers. *arXiv preprint arXiv:2503.01805*, 2025.

[30] 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? *Advances in neural information processing systems*, 34:28877–28888, 2021.

[31] Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xiaochuang Han, Tianxing He, and Yulia Tsvetkov. Can llm graph reasoning generalize beyond pattern memorization? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2289–2305, 2024.

[32] Qi Zhu, Da Zheng, Xiang Song, Shichang Zhang, Bowen Jin, Yizhou Sun, and George Karypis. Parameter-efficient tuning large language models for graph representation learning. *arXiv preprint arXiv:2404.18271*, 2024.

[33] Wenhao Zhu, Tianyu Wen, Guojie Song, Liang Wang, and Bo Zheng. On structural expressive power of graph transformers. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3628–3637, 2023.

[34] Wenhao Zhu, Guojie Song, Liang Wang, and Shaoguo Liu. Anchorgt: Efficient and flexible attention architecture for scalable graph transformers. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 5707–5715. ijcai.org, 2024.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer:[Yes]

   Justification:[Yes] We have claimed the scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer:[Yes]

   Justification:[Yes] We have discussed limitation

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: [Yes] We have the assumptions and proofs

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: [Yes] We have the experimental results

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: [Yes] We have the description in the appendix and provide the code in the supplement material

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: [Yes] we've set the experiment details

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: [Yes] Yes, the experiment statistical significant.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: [Yes] We've included it in the appendix

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: [Yes] We've reviewed

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: [Yes] Yes, we've discussed

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: [NA] No such risks

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer:[Yes]

    Justification: [Yes] We've cited

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: [NA] The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: [NA] The paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: [NA] the paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: [Yes] we only use LLMs for writing and editing

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Table 5: Comparison among GNNs, Graph Transformers, and Decoder-Only Transformers (LLMs).

	GNNs	Graph Transformer	Decoder-Only Transformer
Input	Discrete graph data (node features, adjacency matrix)	Discrete graph data (node features, adjacency matrix)	Textual description of a graph
Output	Scalar value (e.g., count) or fixed-size vector (e.g., for classification)	Scalar value (e.g., count) or fixed-size vector (e.g., for classification)	Text tokens forming a human-readable structural description in indefinite length
Learning Formulation	Encode graph via message passing and neighborhood aggregation; supervised on scalar outputs	Encode graph via graph-aware self-attention; supervised on scalar outputs	Next-token prediction over graph-structured text
Mechanism	1-WL [4]	k-WL [14] / SEG-WL [33]	The proposed ISF (Ours)

Appendix

A Boarder Impact

In this work, we present new insights into how Transformers solve the substructure extraction task, offering a deeper understanding of their internal mechanisms. Our contributions span three key areas: (1) we introduce a novel concept of Induced Substructure Filtration (ISF), instructing LLMs in structure data understanding; (2) we propose a decomposition-based approach for tackling complex substructure reasoning by breaking down intricate patterns into simpler components, enabling more efficient extraction. This concept of thinking in substructures can generalize beyond graph tasks—for example, complementing step-by-step reasoning with pattern-by-pattern thinking; and (3) we provide both theoretical and empirical evidence supporting the development of graph foundation models, highlighting the potential of Transformers as backbones for structured learning tasks.

B Limitation

In this work, we focus primarily on the fundamentals of the substructure extraction task. However, other substructure-related tasks remain to be explored in future research. Additionally, our current study provides a high-level overview of decoder-only Transformers, leaving out theoretical details. Future work can extend this foundation to develop a more comprehensive and rigorous understanding.

C Related work

The mechanisms by which machine learning models learn graph-related problems have been widely studied, ranging from graph neural networks to transformers. However, due to their differing input–output formulations, the underlying mechanisms vary substantially, as summarized in Table 5.

D More on Theoretical Analysis

D.1 Preliminaries

Let $G = (V, E)$ be a directed graph, where $V = \{v_1, \ldots, v_n\}$. For each vertex $v_i \in V$, denote $N(v_i) = \{v \in V \mid (v_i, v) \in E\} = \{v_i^1, \cdots, v_i^{m_i}\}$ as the set of its neighbors. We formally define two sequence representations of the graph G where each vertex identifier (v_i, v_i^j) and the special symbols (":"; ","; "|") are treated as individual tokens.

The adjacency list sequence representation $\mathsf{AL}(G)$ is constructed by concatenating blocks of tokens for each vertex v_i, separated by special token ",". The block for vertex v_i consists of the token v_i and token ":", followed by the sequence of tokens representing its neighbors $v_i^1, \ldots, v_i^{m_i}$. Formally, we have the following definition.

Definition D.1 (Adjacency List Graph Representation)**.** For a directed graph $G = (V, E)$ with $V = \{v_1, \cdots, v_n\}$, denote $N(v_i) = \{v \in V \mid (v_i, v) \in E\} = \{v_i^1, \cdots, v_i^{m_i}\}$ as the set of its

neighbors. The adjacency list graph representation of G is defined as

$$\mathsf{AL}(G) = (v_1; \text{“:”}; v_1^1; \cdots; v_1^{m_1}; \text{“,”}; \cdots; \text{“,”}; v_n; \text{“:”}; v_n^1; \cdots; v_n^{m_n}).$$

The edge list sequence representation $\mathsf{EL}(G)$ is constructed by sequentially listing token pairs (v_i, v_i^j) representing edges, separated by the “|” token. We give the formal definition below.

Definition D.2 (Edge List Graph Representation)**.** For a directed graph $G = (V, E)$ with $V = \{v_1, \cdots, v_n\}$, denote $N(v_i) = \{v \in V \mid (v_i, v) \in E\} = \{v_i^1, \cdots, v_i^{m_i}\}$ as the set of its neighbors. The edge list graph representation of G is defined as

$$\mathsf{EL}(G) = (v_1; v_1^1; \text{“|”}; \cdots; \text{“|”}; v_1; v_1^{m_1}; \text{“|”}; \cdots; \text{“|”}; v_n; v_n^1; \text{“|”}; \cdots; \text{“|”}; v_n; v_n^{m_n}).$$

We also define adjacency matrix for graph G as $A(G)$, where

$$A(G)_{i,j} = \begin{cases} 1, & (v_i, v_j) \in E \\ 0, & (v_i, v_j) \notin E \end{cases}.$$

In the subsequent analysis, we need a vectorized representation of a matrix or tensor. We first define tensor vectorization as follows.

Definition D.3 (Tensor Vectorization)**.** Let $\mathcal{A}$ be a d-dimensional tensor (or tensor of order d) with dimensions $(n_1, n_2, \ldots, n_d)$, denoted as $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$. The elements of $\mathcal{A}$ are indexed by a tuple $(i_1, i_2, \ldots, i_d)$, where $1 \le i_k \le n_k$ for $k \in \{1, 2, \ldots, d\}$.

The vectorization of $\mathcal{A}$, denoted as $\mathsf{vec}(\mathcal{A})$, is a one-dimension vector $\mathbf{a} \in \mathbb{R}^N$, where $N = \prod_{k=1}^d n_k$ is the total number of elements in $\mathcal{A}$.

The elements of the vector $\mathbf{a} = (a_1, a_2, \ldots, a_N)$ are obtained by arranging the elements $\mathcal{A}_{i_1,i_2,\ldots,i_d}$ of the tensor $\mathcal{A}$ in row-major order. Specifically, the tensor element $\mathcal{A}_{i_1,i_2,\ldots,i_d}$ maps to the vector element a_j, where the index j ($1 \le j \le N$) is determined by the following formula:

$$j = 1 + \sum_{m=1}^{d} \left((i_m - 1) \prod_{l=m+1}^{d} n_l \right).$$

Here, the empty product convention is used, i.e., $\prod_{l=d+1}^{d} n_l \triangleq 1$.

Remark D.4. This indexing scheme corresponds to ordering the elements such that the last index i_d varies the fastest, followed by the second-to-last index i_{d-1}, and so on, with the first index i_1 varying the slowest. For example, in the case of a matrix ($d = 2$), this corresponds to concatenating the rows of the matrix.

Definition D.5 (Subgraph Isomorphism Indicator Tensor)**.** Let $G = (V, E)$ and $G' = (V', E')$ be two directed graphs, referred to as the target graph and the query graph, respectively. Let $n = |V|$ and $k = |V'|$, and denote $V = \{v_1, v_2, \ldots, v_n\}$, $V' = (v_1', v_2', \ldots, v_k')$.

The subgraph isomorphism indicator tensor $\mathcal{T}(G, G')$ associated with G, G', and the chosen vertex orderings is a k-dimensional tensor of size $n \times n \times \cdots \times n$. An element $\mathcal{T}(G, G')_{j_1,j_2,\ldots,j_k}$ of the tensor $\mathcal{T}(G, G')$, indexed by a tuple $(j_1, j_2, \ldots, j_k)$ where $1 \le j_l \le n$ for all $l \in \{1, \ldots, k\}$, satisfies:

$$\mathcal{T}(G, G')_{j_1,j_2,\ldots,j_k} \begin{cases} = 1, & \text{if the mapping } f : V' \to V \text{ defined by } f(v_l') = v_{j_l} \text{ for } l = 1, \ldots, k \\ & \text{satisfies both conditions:} \\ & \quad \text{(i) Injectivity: } v_{j_1}, v_{j_2}, \ldots, v_{j_k} \text{ are distinct vertices in } V \\ & \quad\quad \text{(i.e., } j_l \ne j_m \text{ for all } 1 \le l < m \le k). \\ & \quad \text{(ii) Edge Preservation: For every directed edge } (v_p', v_q') \in E', \\ & \quad\quad \text{the directed edge } (f(v_p'), f(v_q')) = (v_{j_p}, v_{j_q}) \text{ exists in } E. \\ \le 0, & \text{otherwise.} \end{cases}$$

That is, $\mathcal{T}(G, G')_{j_1,\ldots,j_k} = 1$ if and only if the sequence of target vertices $(v_{j_1}, \ldots, v_{j_k})$ forms a subgraph in G that is isomorphic to G' under the mapping implied by the indices and the fixed vertex orderings.

Throughout our theoretical analysis, we consider log-precision auto-regressive Transformer, instead of constant-precision Transformer. See [12, Appendix B] for more discussions.

D.2 Technical Lemmas

Lemma D.6 (Adjacency Matrix Extraction)**.**

(i) For any integer n, there exists a two-layer log-precision Transformer with single attention head and hidden dimension $O(n^2)$, such that for any directed graph $G = (V, E)$ with $|V| = n$, the Transformer can output $\mathsf{vec}(A(G))$ for input sequence $\mathsf{AL}(G)$.

(ii) For any integer n, there exists a two-layer log-precision Transformer with single attention head and hidden dimension $O(n^2)$, such that for any directed graph $G = (V, E)$ with $|V| = n$, the Transformer can output $\mathsf{vec}(A(G))$ for input sequence $\mathsf{EL}(G)$.

Proof. We need token embeddings to encode the index of the node (i for vertex v_i), the type of the node (v_i or v_i^j), and absolute positional embedding.

The first attention layer finds all the edges in G. For adjacency list graph representation, we COPY the value of $n \times (i-1)$ (from the position v_i) to the positions $v_i^1, \cdots, v_i^{m_i}$. Applying [11, Lemma C.7] and setting $\langle \boldsymbol{q}_i, \boldsymbol{k}_j \rangle = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2$, where $\boldsymbol{x}_i$ is the type of the node in the embedding suffices. For edge list graph representation, it suffices to COPY from the previous token. Thus we can set $\langle \boldsymbol{q}_i, \boldsymbol{k}_j \rangle = (i-j-1)^2$.

The subsequent MLP calculates $\mathsf{vec}(A(G[\{v_i, v_j\}]))$ for each edge. Notice that the value on index k can be formulated as

$$\begin{aligned} \mathbf{1}_{k=n\times(i-1)+j} = \text{ReLU}[k - n \times (i-1) - j + 1] + \text{ReLU}[k - n \times (i-1) - j - 1] \\ - 2\text{ReLU}[k - n \times (i-1) - j]. \end{aligned} \tag{1}$$

By [11, Lemma C.2], Equation (1) can be calculated with constant hidden dimension.

The second attention layer aggregates all $\mathsf{vec}(A(G[\{v_i, v_j\}]))$ for each edge. We first calculate the MEAN of all valid $\mathsf{vec}(A(G[\{v_i, v_j\}]))$ from the last layer by [11, Lemma C.8]. The result can be expressed as $\mathsf{vec}(A'(G))$ where

$$A'(G)_{i,j} \begin{cases} \geq \frac{1}{n^2}, & (v_i, v_j) \in E \\ = 0, & (v_i, v_j) \notin E \end{cases}.$$

Thus we can get $A(G)_{i,j}$ by

$$A(G)_{i,j} = n^2 \cdot \text{ReLU}\left(\frac{1}{n^2} - A'(G)_{i,j}\right),$$

which can be implemented in the subsequent MLP by [11, Lemma C.2]. □

Lemma D.7 (One-Step Subgraph Isomorphism Indicator Tensor Calculation)**.** *Fix integers $n, k, m \geq 1$. Let $V' = \{v_1', \ldots, v_k'\}$ be a set of k vertices, and let $\mathcal{F}(V') = (V_1', V_2', \ldots, V_m')$ be a k-node m-filtration on V' (defined in Definition 3.1). There exists m two-layer MLPs with GeLU activation $\boldsymbol{f}_1, \cdots, \boldsymbol{f}_m$ each with hidden dimension $O(n^2k^2)$, such that for any directed graphs $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $G' = (V', E')$, $\boldsymbol{f}_i$ can output $\mathsf{vec}(\mathcal{T}(G, G'[V_i']))$ for input $\mathsf{vec}(A(G)) \oplus \mathsf{vec}(A(G')) \oplus \mathsf{vec}(\mathcal{T}(G, G'[V_{i-1}']))$ ($\oplus$ denotes concatenation), where $V_0' := \varnothing$.*

Proof. Denote $k_i = |V_i'|$. Without loss of generosity, we assume $V_i' = \{v_1, \cdots, v_{k_i}'\}$. Notice that $\mathcal{T}(G, G_i')_{j_1, \cdots, j_{k_i}}$ can be calculated as

$$\begin{aligned} \mathcal{T}(G, G_{i-1}')_{j_1, \cdots, j_{k_{i-1}}} + \sum_{x,y \leq k_i;\ x > k_{i-1} \vee y > k_{i-1}} \left[A(G')_{x,y} A(G)_{j_x, j_y} - A(G')_{x,y}\right] \\ - \mathbf{1}_{\exists 1 \leq x < y \leq k_i,\ j_x = j_y}. \end{aligned}$$

Thus it suffices to calculate the value of $A(G')_{x,y} A(G)_{x',y'}$ for $1 \leq x, y \leq k,\ 1 \leq x', y' \leq n$. By [11, Lemma C.1], this can be implemented with $O(n^2k^2)$ hidden dimension. □

D.3 Proofs for Section 3

Theorem 3.3 (Expressiveness for Progressive Identification)**.** *Given a k-node m-filtration $\mathcal{F}(V')$ on $V' = \{v_1', \ldots, v_k'\}$. For any directed graphs $G = (V, E)$ ($|V| = n$) and $G' = (V', E')$, a*

log-precision Transformer with $m+2$ layers, constant heads, and $O(n^k)$ hidden dimension can output $\mathsf{vec}(\mathcal{T}(G, G'[V_i']))$ *at layer $i+2$ for $i \in \{1, \ldots, m\}$.*

Theorem D.8 (Formal Statement of Theorem 3.3)**.** *Fix integers $n \geq k \geq 1, m \geq 1$. Let $V' = \{v_1', \ldots, v_k'\}$ be a set of k vertices, and let $\mathcal{F}(V') = (V_1', V_2', \ldots, V_m')$ be a k-node m-filtration on V'. There exists a log-precision Transformer with $m+2$ layers, constant number of attention heads, and $O(n^k)$ hidden dimension, such that for any directed graphs $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $G' = (V', E')$, the Transformer processing G, G' can output* $\mathsf{vec}(\mathcal{T}(G, G'[V_i']))$ *at layer $i+2$ for $i \in \{1, \cdots, m\}$. Here,* $\mathsf{vec}(\cdot)$ *is the vectorization for a tensor (formal defined in Definition D.3).*

Proof. The first two layers of Transformer calculates $\mathsf{vec}(A(G))$ and $\mathsf{vec}(A(G'))$ for graph G, G'. The desired output is at the last token for each graph, respectively. By Lemma D.6, this can be implemented with $O(n^2)$ hidden dimension, regardless of the representation of G, G'.

For the next m layers, we first apply [11, Lemma C.7] to COPY $\mathsf{vec}(A(G)), \mathsf{vec}(A(G'))$ in the attention layer. This can be implemented by adding special marks on the last token for each graph. In the subsequent MLP layer, we apply Lemma D.7 to calculate desired results with $O(n^2k^2)$ hidden dimension. □

Theorem 3.5 (Expressiveness for Pattern Extraction)**.** *Under Assumption 3.4, for directed graphs $G = (V, E)$ ($|V| = n$) and $G' = (V', E')$ ($|V'| = k$), a log-precision Transformer with constant depth, constant heads, and $O(n^k)$ hidden dimension can output the unique k-tuple of vertices $(v_{i_1}, \ldots, v_{i_k})$ for which $\mathcal{T}(G, G')_{i_1,\ldots,i_k} = 1$.*

Theorem D.9 (Formal Statement of Theorem 3.5)**.** *Fix integers $n \geq k \geq 1$, and let $V' = \{v_1', \ldots, v_k'\}$. There exists a log-precision Transformer with constant depth, constant number of attention heads, and $O(n^k)$ hidden dimension, such that for any directed graphs $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $G' = (V', E')$ satisfying Assumption 3.4, the Transformer processing G, G' can output the unique tuple $(v_{i_1}, \ldots, v_{i_k})$ for which $\mathcal{T}(G, G')_{i_1,\cdots,i_k} = 1$.*

Proof. We first take $m = 1$ in Theorem 3.3, indicating that a constant depth Transformer can output $\mathsf{vec}(\mathcal{T}(G, G'))$ for any directed graphs G, G'. By Assumption 3.4, $\mathrm{ReLU}(\mathsf{vec}(\mathcal{T}(G, G')))$ is a one-hot vector and can be obtained with a two-layer MLP via [11, Lemma C.2].

Notice that

$$i_x = \sum_{1 \leq i_1, \cdots, i_k \leq n} i_x \cdot \mathrm{ReLU}(\mathsf{vec}(\mathcal{T}(G, G'))),$$

thus by linear transformation we can obtain $(i_1, \cdots, i_k)$ for which $\mathcal{T}(G, G')_{i_1,\cdots,i_k} = 1$, from the corresponding one-hot vector $\mathrm{ReLU}(\mathsf{vec}(\mathcal{T}(G, G')))$.

The final step is to output $(v_{i_1}, \ldots, v_{i_k})$ sequentially. This can be obtained by adding one-hot positional encoding in all the output tokens to determine the current output position. Therefore, the next token can be obtained by calculating the inner product between $(i_1, \cdots, i_k)$ and the positional encoding. Since all the steps can be finished in constant layers, we finished our proof. □

Theorem 3.8 (Expressiveness for Single-Shape-Multi-Num Extraction)**.** *Fix integers $n \geq k \geq 1$. There exists a log-precision Transformer with constant depth, constant number of attention heads, and $O(n^k)$ hidden dimension that can complete Single-Shape-Multi-Num Extraction defined in Definition 3.7 for directed graphs $G = (V, E)$ ($|V| = n$) and $G' = (V', E')$ ($|V'| = k$).*

Theorem D.10 (Formal Statement of Theorem 3.8)**.** *Fix integers $n \geq k \geq 1$, and let $V' = \{v_1', \ldots, v_k'\}$. There exists a log-precision Transformer with constant depth, constant number of attention heads, and $O(n^k)$ hidden dimension, such that for any directed graphs $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $G' = (V', E')$, the Transformer processing G, G' can output all the tuples $(v_{i_1}, \ldots, v_{i_k})$ for which $\mathcal{T}(G, G')_{i_1,\cdots,i_k} = 1$.*

Proof. Follow the proof of Theorem 3.5, we first calculate $\boldsymbol{v}^1 = \mathrm{ReLU}(\mathsf{vec}(\mathcal{T}(G, G')))$ which marks all feasible tuples with 1, and 0 otherwise.

Next, we calculate $\boldsymbol{v}^2$ where $\boldsymbol{v}_i^2 = \boldsymbol{v}_1^1 + \cdots + \boldsymbol{v}_i^1$ by linear projection, and define $\boldsymbol{v}_i^3 = \boldsymbol{v}_i^1 \boldsymbol{v}_i^2$. In $\boldsymbol{v}_i^3$, the feasible tuples are marked as $1, 2, \cdots$, and 0 otherwise. By [11, Lemma C.1], $\boldsymbol{v}^2, \boldsymbol{v}^3$ can be obtained with a two-layer MLP.

The final step is to determine which position in which tuple the next-token corresponds to. This can be obtained by adding special positional encoding in the outputs. When the Transformer need to output the x-th tuple, it can first obtain the corresponding one-hot vector by the following formula:

$$\text{ReLU}(\boldsymbol{v}^3 - x - 1) + \text{ReLU}(\boldsymbol{v}^3 - x + 1) - 2 \cdot \text{ReLU}(\boldsymbol{v}^3 - x),$$

then following with the proofs in Theorem 3.5 to output the current position. By [11, Lemma C.2], the above steps can be obtained with constant-layer MLPs, which concludes our proof. □

Theorem 3.10 (Expressiveness for Multi-Shape-Single-Num Extraction)**.** *Fix integers $n \geq k \geq 1$. There exists a log-precision Transformer with constant depth, constant heads, and $O(n^k)$ hidden dimension that can complete Multi-Shape-Single-Num Extraction defined in Definition 3.9 for a directed graph $G = (V, E)$ ($|V| = n$) and any target subgraph $G' = (V', E')$ with $|V'| = k' \leq k$ satisfying Assumption 3.4.*

Theorem D.11 (Formal Statement of Theorem 3.10)**.** *Fix integers $n \geq k \geq 1$. There exists a log-precision Transformer with constant depth, constant number of attention heads, and $O(n^k)$ hidden dimension, such that for any directed graph $G = (V, E)$ (with $V = \{v_1, \ldots, v_n\}$) and $G' = (V', E')$ (with $V' = \{v'_1, \cdots, v'_{k'}\}$ where $k' \leq k$) satisfying Assumption 3.4, the Transformer processing G, G' can output the unique tuples $(v_{i_1}, \ldots, v_{i_{k'}})$ for which $\mathcal{T}(G, G')_{i_1, \cdots, i_{k'}} = 1$.*

Proof. The proof is based on that of Theorem 3.5, and we extend G' to $\hat{G}'$ with $k - k'$ extra isolated node.

Now, for more general case that $k' \leq k$, there may exist multiple tuples $(v_{i_1}, \cdots, v_{i_k})$ such that $\mathcal{T}(G, \hat{G}')_{i_1, \cdots, i_k} = 1$. However, by Assumption 3.4, all these tuples shares the same $i_1, \cdots, i_{k'}$. Therefore, we can first obtain a one-hot vector via

$$\text{ReLU}(\boldsymbol{v}^3) + \text{ReLU}(\boldsymbol{v}^3 - 2) - 2 \cdot \text{ReLU}(\boldsymbol{v}^3 - 1),$$

where $\boldsymbol{v}^3$ is defined in the proof of Theorem 3.8. Finally, it suffices to output the corresponding $(v_{i_1}, \cdots, v_{i'_k})$, which is similar to the final step of Theorem 3.5. □

D.4 Theoretical Results for Section 5.1

Assumption D.12. For directed graphs $G = (V, E)$ with $V = \{v_1, \cdots, v_n\}$, $G' = (V', E')$ with $V' = \{v'_1, \cdots, v'_k\}$ and $V'_1, \cdots, V'_t \subseteq V'$, and a collection of t vertex subsets $V'_1, \ldots, V'_t \subseteq V'$. It is assumed that:

(i) There exists a *unique* k-tuple of distinct vertex indices $(i_1, \ldots, i_k)$ from $\{1, \ldots, n\}$ such that $\mathcal{T}(G, G')_{i_1, \ldots, i_k} = 1$.

(ii) For each $j \in \{1, \ldots, t\}$, there is a fixed constant $c \geq 1$ such that the number of distinct $|V'_j|$-tuples of distinct vertex indices $(i_1, \ldots, i_{|V'_j|})$ from $\{1, \ldots, n\}$ for which $\mathcal{T}(G, G'[V'_j])_{i_1, \ldots, i_{|V'_j|}} = 1$ is at most c.

Theorem D.13. *Fix integers $n \geq k \geq 1$ and $t \geq 1$. Let $G' = (V', E')$ be a fixed directed graph with $V' = \{v'_1, \ldots, v'_k\}$. Let $V'_1, \ldots, V'_t$ be a collection of subsets of V' such that G' is covered by the subgraphs induced by these subsets, meaning $V' = \bigcup_{j=1}^t V'_j$ and $E' \subseteq \bigcup_{j=1}^t E(G'[V'_j])$. Denote $q = \max_{j \in \{1, \ldots, t\}} |V'_j|$.*

There exists a log-precision Transformer with constant depth, constant number of attention heads, and $O(n^q + c^t + c^2 t^2 n)$ hidden dimension, such that: For any directed graph $G = (V, E)$ (with $V = \{v_1, \ldots, v_n\}$) that, together with the predefined G' and subsets $V'_1, \ldots, V'_t$, satisfies Assumption D.12 (where c is the constant from Assumption D.12), the Transformer processing G can

(i) First, for each $j = 1, \ldots, t$, output a special token $\langle S_j \rangle$, then identify and output all distinct $|V'_j|$-tuples of vertices $(v_{i_1}, \ldots, v_{i_{|V'_j|}})$ from G such that $\mathcal{T}(G, G'[V'_j])_{i_1, \ldots, i_{|V'_j|}} = 1$.

(ii) Subsequently, output a special token $\langle \text{ANS} \rangle$ and the unique k-tuple of vertices $(v_{i_1}, \ldots, v_{i_k})$ from G such that $\mathcal{T}(G, G')_{i_1, \ldots, i_k} = 1$.

Remark D.14. If we assume c, t are both constants, then the result becomes $O(n^q)$, which demonstrates the advantages of thinking in substructures.

Remark D.15. Theorem D.13 highlights a trade-off concerning the hidden dimension complexity, $O(n^q + c^t + c^2t^2n)$. This complexity is influenced by:

- t: the number of intermediate decomposition steps, or the CoT steps.
- q: the maximum size of any intermediate subgraph $G'[V'_j]$ considered during these steps.
- c: the maximum number of instances (matches) in G for any such intermediate subgraph $G'[V'_j]$.

When t increases (employing more, potentially smaller, intermediate steps), q generally decreases. However, c may increase, as simpler or smaller intermediate subgraphs could appear more frequently. In this scenario, the n^q component of the hidden dimension tends to decrease, while the $c^t + c^2t^2n$ components are likely to increase.

Conversely, when t decreases (employing fewer, potentially larger, intermediate steps), q generally increases. Correspondingly, c may decrease, as more complex or larger intermediate subgraphs could be less common. This tends to increase the n^q component, while the $c^t + c^2t^2n$ components are likely to decrease.

Thus, the optimal decomposition strategy for minimizing the required hidden dimension depends on the interplay between these parameters, dictated by the specific problem structure.

Proof. We first design the necessary embeddings.

1. For the special token $\langle S_j \rangle$, we need a q^2 dimension vector representing $\mathsf{vec}(A(G[V'_j])$ (if $|V'_j| < q$, then we need to add $q - |V'_j|$ isolated nodes); and a n^q dimension vector representing $\mathsf{vec}(\mathcal{T}^{(j)})$ where $\mathcal{T}^{(j)}$ is a q-dimensional tensor of size $n \times n \times \cdots \times n$ defined as:
$$\mathcal{T}^{(j)}_{i_1,\cdots,i_q} = \begin{cases} 0, & \forall 1 \le x < y \le |V'_j|,\ i_x \neq i_y;\ \forall |V'_j| < x \le q,\ i_x = 1 \\ 1, & \text{otherwise} \end{cases}. \tag{2}$$
2. For the output answer tokens in step (i), we need a ctn dimension vector. For v_{i_x} in the y-th tuple for the subgraph induced by V'_j, the embedding satisfies: the value on the $cn(j-1)+x'$ dimension is i_x, while the others are 0. Here, x' is the node v_{i_x} corresponds to in origin G' ($v'_{x'}$).

To get the desired output sequence, we need to complete the following tasks:

- **Task 1:** At the position of $\langle S_j \rangle$, we need to get all the tuples $(i_1, \ldots, i_{|V'_j|})$ for which $\mathcal{T}(G, G'[V'_j])_{i_1,\ldots,i_{|V'_j|}} = 1$.
- **Task 2:** At the position of $\langle \text{ANS} \rangle$, we need to get the *unique* tuple $(i_1, \ldots, i_k)$ for which $\mathcal{T}(G, G')_{i_1,\ldots,i_k} = 1$.

For task 1, the idea is similar to the proof of Theorem 3.8. We first use Lemma D.6 to extract $\mathsf{vec}(A(G))$ for input graph G, then apply [11, Lemma C.7] to COPY $\mathsf{vec}(A(G))$ to the current position. Next, we calculate $\mathsf{vec}(\mathcal{T}'(G, \hat{G}'[V'_j]))$. Here, $\hat{G}'[V'_j]$ is obtained by adding $q - |V'_j|$ isolated nodes on $G'[V'_j]$; and

$$\mathcal{T}'\left(G, \hat{G}'[V'_j]\right)_{i_1,\cdots,i_q} \begin{cases} = 1, & \text{if } \mathcal{T}\left(G, G'[V'_j]\right)_{i_1,\ldots,i_{|V'_j|}} = 1 \text{ and } i_{|V'_j|+1} = \cdots = i_q = 1 \\ \le 0, & \text{otherwise} \end{cases}.$$

Thus, $\mathsf{vec}(\mathcal{T}'(G, \hat{G}'[V'_j]))$ is a n^q dimensional vector. Notice that $\mathcal{T}'(G, \hat{G}'[V'_j])_{i_1,\cdots,i_q}$ can be calculated as

$$\sum_{1 \le x,y \le q} \left[A\left(\hat{G}'[V'_j]\right)_{x,y} A(G)_{i_x,i_y} - A\left(\hat{G}'[V'_j]\right)_{x,y} \right] - \mathcal{T}^{(j)}_{i_1,\cdots,i_q},$$

where $\mathcal{T}^{(j)}$ is defined in Equation (2). The following steps are similar to the proof of Theorem 3.8, while the only difference is we only want the first $|V_j'|$ dimension. This can be implemented by modifying the positional encoding to give the correct position for the next-token.

For task 2, we first aggregate all the previous $|V_j'|$-tuples for $j = 1, \cdots, t$ using MEAN operation in [11, Lemma C.8]. We then multiplies the result with the sequence length (which can be obtained by absolute positional encoding). After this, we get a ctn-dimension vector $(\boldsymbol{b}_{1,1}, \cdots, \boldsymbol{b}_{1,c}, \boldsymbol{b}_{2,1}, \cdots, \boldsymbol{b}_{t,1}, \cdots, \boldsymbol{b}_{t,c})$. Here, $\boldsymbol{b}_{i,j}$ is a n-dimension vector corresponds to the j-th tuple for the subgraph induced by V_i'.

Next, we maintain a t-dimension tensor $\mathcal{T}^{\text{ans}}$ of size $c \times c \times \cdots \times c$, defined as

$$\mathcal{T}^{\text{ans}}_{i_1,\cdots,i_t} \begin{cases} = 0, & \text{if } \boldsymbol{b}_{1,i_1}, \cdots, \boldsymbol{b}_{t,i_t} \text{ can be combined as } G' \\ \geq 1, & \text{otherwise} \end{cases}.$$

By Assumption D.12, there exists a *unique* t-tuple $(i_1, \ldots, i_t)$ such that $\mathcal{T}^{\text{ans}}_{i_1,\cdots,i_t} = 0$. Notice that $\boldsymbol{b}_{1,i_1}, \cdots, \boldsymbol{b}_{t,i_t}$ can be combined as G' if and only if the following holds:

$$\begin{cases} \forall 1 \leq x \leq t,\ \boldsymbol{b}_{x,i_x} \neq \boldsymbol{0} \\ \forall 1 \leq x < y \leq t,\ \forall 1 \leq z \leq n,\ (\boldsymbol{b}_{x,i_x})_z = (\boldsymbol{b}_{y,i_y})_z \text{ or } (\boldsymbol{b}_{x,i_x})_z = 0 \text{ or } (\boldsymbol{b}_{y,i_y})_z = 0 \end{cases} \tag{3}$$

Since $(\boldsymbol{b}_{y,i_y})_z \in \{0, 1, \cdots, n\}$, Equation (3) is equivalent to

$$\begin{cases} \forall 1 \leq x \leq t,\ \text{ReLU}\left[1 - \sum_{1 \leq z \leq n} (\boldsymbol{b}_{x,i_x})_z\right] = 0 \\ \forall 1 \leq x < y \leq t,\ \forall 1 \leq z \leq n,\ \text{ReLU}[(\boldsymbol{b}_{x,i_x})_z - (\boldsymbol{b}_{y,i_y})_z] + \text{ReLU}[(\boldsymbol{b}_{y,i_y})_z - (\boldsymbol{b}_{x,i_x})_z] = 0 \\ \qquad\qquad \text{or } (\boldsymbol{b}_{x,i_x})_z = 0 \text{ or } (\boldsymbol{b}_{y,i_y})_z = 0 \end{cases}$$

The second condition is equivalent to $\forall 1 \leq x < y \leq t,\ \forall 1 \leq z \leq n$,

$$\begin{aligned} &\text{ReLU}\left[1 - \text{ReLU}[(\boldsymbol{b}_{x,i_x})_z - (\boldsymbol{b}_{y,i_y})_z] - \text{ReLU}[(\boldsymbol{b}_{y,i_y})_z - (\boldsymbol{b}_{x,i_x})_z]\right] \\ &+ \text{ReLU}[1 - (\boldsymbol{b}_{x,i_x})_z] + \text{ReLU}[1 - (\boldsymbol{b}_{y,i_y})_z] \geq 1, \end{aligned}$$

or

$$\begin{aligned} &\text{ReLU}\left[1 - \text{ReLU}\left[1 - \text{ReLU}[(\boldsymbol{b}_{x,i_x})_z - (\boldsymbol{b}_{y,i_y})_z] - \text{ReLU}[(\boldsymbol{b}_{y,i_y})_z - (\boldsymbol{b}_{x,i_x})_z]\right]\right. \\ &\left. -\text{ReLU}[1 - (\boldsymbol{b}_{x,i_x})_z] - \text{ReLU}[1 - (\boldsymbol{b}_{y,i_y})_z]\right] = 0. \end{aligned}$$

Thus, for any tuple $(i_t, \cdots, i_t)$, we can get $\mathcal{T}^{\text{ans}}_{i_1,\cdots,i_t}$ via an MLP with constant layers and $O(nt^2)$ hidden dimension. Notice that there are many components remaining the same when calculating different $\mathcal{T}^{\text{ans}}_{i_1,\cdots,i_t}$. We can calculate

$$\begin{aligned} &\text{ReLU}\left[1 - \text{ReLU}\left[1 - \text{ReLU}[(\boldsymbol{b}_{p_1,q_1})_z - (\boldsymbol{b}_{p_2,q_2})_z] - \text{ReLU}[(\boldsymbol{b}_{p_2,q_2})_z - (\boldsymbol{b}_{p_1,q_1})_z]\right]\right. \\ &\left. -\text{ReLU}[1 - (\boldsymbol{b}_{p_1,q_1})_z] - \text{ReLU}[1 - (\boldsymbol{b}_{p_2,q_2})_z]\right] \end{aligned}$$

for all $(p_1, q_1), (p_2, q_2)$ pairs and $1 \leq z \leq n$, which are $O(c^2t^2n)$. Each can be calculated via an MLP with constant depth and constant hidden dimension.

Finally, we will calculate the *unique* t-tuple $(i_1, \ldots, i_t)$ such that $\mathcal{T}^{\text{ans}}_{i_1,\cdots,i_t} = 0$. Notice that

$$i_x = \sum_{1 \leq i_1, \cdots, i_t \leq c} \text{ReLU}(1 - \mathcal{T}^{\text{ans}}_{i_1,\cdots,i_t}) \cdot \left(\sum_{1 \leq j \leq t} \boldsymbol{b}_{j,i_j}\right),$$

which can be calculated via an MLP with constant depth and $O(c^t)$ hidden dimension by [11, Lemma C.1]. □

D.5 Theoretical Results for Section 5.2

Theorem D.16. *Fix integers $n \geq k \geq 1$, and let $V = \{v_1, \cdots, v_n\}, V' = \{v_1', \cdots, v_k'\}$. Fix a feature function $\varphi : V \cup V' \to \mathbb{Z}$. There exists a log-precision Transformer with constant depth, constant number of attention heads, and $O(n^k)$ hidden dimension, such that for any directed*

Table 6: Hyperparameter details

heads	embedding	drop out rate	batch size	learning rate	max epoch
12	384	0.2	2048	0.001	40000

graphs $G = (V, E)$ and $G' = (V', E')$, the Transformer processing G, G' can output all the tuples $(v_{i_1}, \cdots, v_{i_k})$ that satisfy both of the following conditions:

(i) Subgraph Isomorphism: The subgraph of G induced by the set of vertices $\{v_{i_1}, \cdots, v_{i_k}\}$ is isomorphic to G' under the mapping $v'_p \mapsto v_{i_p}$ for $p \in \{1, \cdots, k\}$. That is, $\mathcal{T}(G, G')_{i_1,\cdots,i_k} = 1$.

(ii) Feature Matching: For all $p \in \{1, \ldots, k\}$, the feature of the p-th vertex in the tuple from G matches the feature of the p-th vertex in V', i.e., $\varphi(v_{i_p}) = \varphi(v'_p)$.

Proof. The proof is similar to that of Theorem 3.8. We define a k-dimensional tensor of size $n \times n \times \cdots \times n$ $\mathcal{T}'(G, G')$ as

$$\mathcal{T}'(G, G')_{j_1,\cdots,j_k} \begin{cases} = 1, & \text{if the mapping } f : V' \to V \text{ defined by } f(v'_l) = v_{j_l} \text{ for } l = 1, \ldots, k \\ & \text{satisfies both conditions:} \\ & \quad \text{(i) Injectivity: } v_{j_1}, v_{j_2}, \ldots, v_{j_k} \text{ are distinct vertices in } V \\ & \quad\quad \text{(i.e., } j_l \neq j_m \text{ for all } 1 \leq l < m \leq k). \\ & \quad \text{(ii) Edge Preservation: For every directed edge } (v'_p, v'_q) \in E', \\ & \quad\quad \text{the directed edge } (f(v'_p), f(v'_q)) = (v_{j_p}, v_{j_q}) \text{ exists in } E. \\ & \quad \text{(iii) Feature Matching: For all } p \in \{1, \cdots, k\}, \text{ the features of the} \\ & \quad\quad p\text{-th vertex match, i.e., } \varphi(v_{i_p}) = \varphi(v'_p). \\ \leq 0, & \text{otherwise.} \end{cases}$$

Notice that $\mathcal{T}'(G, G')_{j_1,\cdots,j_k}$ can be obtained as

$$\mathcal{T}(G, G')_{j_1,\cdots,j_k} - \sum_{1 \leq x \leq k} \mathbf{1}_{\varphi(v_{i_x}) \neq \varphi(v'_x)},$$

or

$$\mathcal{T}(G, G')_{j_1,\cdots,j_k} - \sum_{1 \leq x \leq k} \left[\text{ReLU}(\varphi(v_{i_x}) - \varphi(v'_x)) + \text{ReLU}(\varphi(v'_x) - \varphi(v_{i_x}))\right].$$

Therefore, it suffices to COPY the feature while constructing the adjacency matrix $A(G), A(G')$. And it suffices to further calculate the value of $\text{ReLU}\left[\varphi(v_i) - \varphi(v'_j)\right]$, $\text{ReLU}\left[\varphi(v'_j) - \varphi(v_i)\right]$, which requires $O(nk)$ hidden dimension in total (by [11, Lemma C.2]). □

E Experiments setting

Here, we provide the details of our experimental setup. We use a lightweight version of the GPT-2 model, which is an implementation version of nano-GPT, with hyperparameters listed in Table 6. 10% of the data is used for validation, and the model is saved when the validation loss reaches its minimum. All experiments are conducted on a machine equipped with 8 NVIDIA A6000 GPUs.e.

E.1 training details in input formulations

We take more 50,000 graphs for training and testing. Each graph contains a target substructure: either a triangle, square, or pentagon. While the number of training samples varies, the test set size remains fixed, as shown in Table 7. Since this is a toy example, we set the Transformer's hidden dimension to a small size of 192.

Table 7: The dataset information for the AL and EL comparison

	#Training data	#Test data	#Node
Triangle	5000	1000	5
Square	15000	1000	8
Pentagon	35000	1000	8

Table 8: Performance across epochs for Square (4 layers) and Pentagon (5 layers).

Epoch	**10000**	**20000**	**30000**	**40000**	**50000**	**60000**
Square (4 layers)						
AL	0.97 ± 0.004	0.98 ± 0.003	–	–	–	–
EL	0.83 ± 0.078	0.93 ± 0.050	0.99 ± 0.006	–	–	–
Pentagon (5 layers)						
AL	0.69 ± 0.017	0.73 ± 0.058	0.84 ± 0.031	0.92 ± 0.004	–	–
EL	0.61 ± 0.017	0.60 ± 0.019	0.74 ± 0.018	0.87 ± 0.010	0.89 ± 0.0056	0.93 ± 0.044

E.2 Multi-Shape setting

To evaluate the discrimination ability of Transformers in detecting multiple structures, we set the evaluations from four perspectives: 1. different numbers of nodes (Triangle vs. Square); 2. the same number of nodes but different numbers of edges (Square vs. Diamond); 3. the same number of nodes and edges, but different edge directions (F-triangle vs. T-triangle); 4. whether the substructure forms a closed loop (Square vs. Path). We construct 600K question-answer pairs to train a 4-layer transformer model. Since triangles require less training data, as suggested in the Multi-num task, we set the training sample ratio of Triangle to Square to 1:6, while maintaining a 1:1 ratio for the other substructure pairs.

E.3 Efficient of EL and AL

EL performs worse than AL when trained for the same number of epochs, but it eventually reaches comparable performance. In our results, we selected the epoch at which AL achieves its best performance. However, we will also provide additional information indicating when EL catches up with AL, as shown in the Table 8 below:

EL with longer input lengths requires more training epochs to achieve the same performance as AL. Although EL and AL are theoretically equivalent in their ability to represent graph structures, the longer input sequences in EL lead to less efficient learning. We will clarify this point in the revision.

E.4 LLMs Experiments

E.4.1 Evaluation on substructure detection

In substructure detection, we set the question prompt as:

Given a structure G, Node 1 is connected to Node 2, 3; Node 2 is connected to.... List all of the square patterns in the graph in the form of: [#1, #2, #3, ...]

Meanwhile, we set the answer as:

The answer is [1, 2, 3]

For the triangle detection task, we use 1,000 training samples and 200 for evaluation. Using supervised fine-tuning (SFT) over 4 epochs, we achieve 58.86% accuracy on the test set. The model responses suggest that LLaMA3.1-8B-Instruct still generates explanatory content, including code, during answer generation.

We also evaluate LLM performance on the square detection task using 283 test samples, which contain only four distinct answer types. As shown in the Table 9, lightweight LLMs fail to extract meaningful patterns without fine-tuning. Due to the high computational cost of training LLMs, we

Table 9: Large Language Models do the ISF process in the middle layers

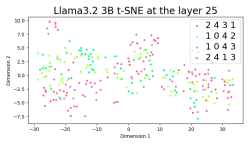
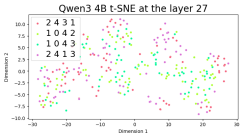
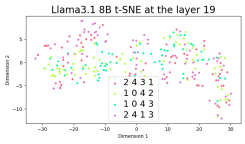
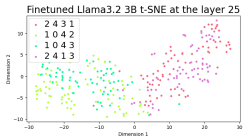
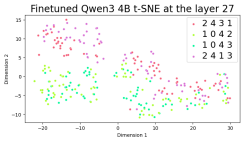
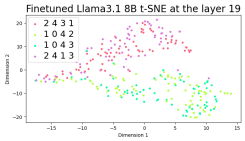
Model	Llama3.2-3B-Instruct	Qwen3-4B-Base	Llama3.1-8B-Instruct
ACC / finetuned ACC	0.0035 / 0.4982	0.0141 / 0.5724	0.0035/0.6572
Vis. for non-finetuned			
Vis. for finetuned			

Table 10: Concept learning

	no-finetune	without topos	with topos
llama	0	0	0.50
Qwen1.5	0	0	0.75
Qwen3B	0	0	0.85

limit the training data to only include samples with these four answer types, using 480 samples for training and 120 for validation. After fine-tuning, we observe a significant improvement in accuracy. Additionally, visualization shows that graphs corresponding to similar answers tend to cluster together.

E.4.2 Evaluation on question prompt

e evaluate how LLMs align conceptual descriptions with underlying topological structures. Specifically, we test LLaMA3.2-3B-Instruct, Qwen2.5-1.5B, and Qwen2.5-3B by setting the temperature to 0.6, running 20 dialogue turns per model, and manually evaluating the responses.

First, we assess whether the models understand the concept of a "house" in graph terminology by prompting them with: *What is house in graphs? Giving the graph in the formulate of: 'Node 1 is connected to nodes 2, 3'* In the baseline setting (without topological prompts), we explicitly teach the concept using natural language definitions generated by Gemini-1.5-Pro. We fine-tune each model using 200 such concept-descriptive sentences. For example, a training sample might look like: *In graph theory, a "house" isn't a standard term like "tree" or "cycle." It usually refers to a specific small graph resembling a house drawing. This graph consists of five vertices and six edges. It's formed by a cycle of four vertices (the "walls" and "floor") with an additional vertex connected to one of the cycle vertices (the "roof peak").* In the train with topos setting, we add a topology description to the house, which is: *The house is described as: G describes an undirected graph among 1, 2, 3, 4, and 5.In this graph: Node 1 is connected to nodes 2, 5.Node 2 is connected to nodes 1, 3, 5. Node 3 is connected to nodes 2, 4. Node 4 is connected to nodes 3, 5. Node 5 is connected to nodes 1, 2, 4.*

As shown in Table 10, the LLMs only learn the topological descriptions training with the terminology terms together. The LLMs do not generate new concepts by the already known knowledge.

E.4.3 Thinking-in-substructures

We use 100K samples for training and 5,000 for testing. Since each composite substructure is composed of different sets of decomposing substructures, the required thinking length varies accordingly. A summary of these decompositions is provided in Table 11

Table 11: Max length for each composite substructure extraction

Substructures	$\lvert\{P_1\}\rvert$	$\lvert\{P_2\}\rvert$	$\lvert\{P_3\}\rvert$	overall length
Diagnoal	95	55	-	150
Diamond	55	95	-	150
House	75	115	-	190
Complex	80	100	110	290

E.4.4 Transformers for moleculars

In this subsection, we introduce the experimental setup for applying transformers to molecular data. Specifically, we focus on the task of functional group recognition, where the goal is to identify the atomic positions corresponding to specific functional groups within molecules. We then introduce the dataset, functional group and experimental dataset construction.

Dataset We conduct experiments on QM9 [16] and PCBA [28]. The QM9 dataset primarily contains quantum mechanical calculated properties of approximately 134,000 molecules, suitable for molecular property prediction and quantum chemistry research. The PCBA dataset, on the other hand, contains activity data for approximately 440,000 molecules against 128 biological assays, making it more suitable for drug screening and bioactivity prediction.

Functional Group We search for molecules containing basic functional groups in the QM9 and PCBA datasets. Specifically, we extract 33,000 molecules containing hydroxyl groups (C-O-H) from QM9, and 13,000 molecules containing carboxyl groups (-COOH) as well as 33,000 molecules containing benzene rings (C_6H_6) from the PCBA dataset. For all these molecules, H atoms are ignored during processing. The maximum number of atoms is 9 for molecules containing hydroxyl groups, while it is 121 for both carboxyl- and benzene-containing molecules.

Experimental Dataset Construction For the hydroxyl group identification task, we first convert molecular graphs into molecular description inputs by omitting H atom. A simple example of such a description is "0 C : 1 O", and the corresponding answer for the position of the C–O–(H) group is "0,1". We then select molecules containing hydroxyl groups, using 30,000 molecules for the training set and 3,000 for the test set.

Similarly, for the identification of molecules containing carboxyl groups and benzene rings, we also convert molecular graphs into molecular description inputs by omitting hydrogen atoms, and generate the corresponding position answers for the target functional groups. We use 10,000 molecules for training and 3,000 for testing in the carboxyl group recognition task. For the benzene ring recognition task, we construct a dataset with 30,000 molecules for training and 3,000 for testing. The maximum input lengths for molecular descriptions are 100, 1000, and 1000 for molecules containing hydroxyl group, carboxyl group, and benzene ring, respectively.

In addition, we construct a mixed dataset containing molecules with hydroxyl and carboxyl groups. Specifically, we use 10,000 hydroxyl-containing molecules and 10,000 carboxyl-containing molecules for training, and 1,500 molecules of each type for testing.