

# Towards Next Graph Token Prediction: Discrete Graph Tokenization for Structural Reasoning in Large Language Models

Zhonghao Wang  
Zhejiang Key Lab of Accessible  
Perception and Intelligent Systems,  
Zhejiang University  
Hangzhou, Zhejiang, China  
wangzhonghao@zju.edu.cn

Yugang Ji  
Alibaba Group  
Hangzhou, Zhejiang, China  
yugang.jyg@alibaba-inc.com

Zhuonan Zheng  
Zhejiang Key Lab of Accessible  
Perception and Intelligent Systems,  
Zhejiang University  
Hangzhou, Zhejiang, China  
zhengzn@zju.edu.cn

Sheng Zhou\*  
Zhejiang Key Lab of Accessible  
Perception and Intelligent Systems,  
Zhejiang University  
Hangzhou, Zhejiang, China  
zhousheng\_zju@zju.edu.cn

Weigao Wen  
Alibaba Group  
Hangzhou, Zhejiang, China  
weigao.wwg@alibaba-inc.com

Minghao Li  
Alibaba Group  
Hangzhou, Zhejiang, China  
liminghao.lmh@alibaba-inc.com

Ming Gu  
Zhejiang Key Lab of Accessible  
Perception and Intelligent Systems,  
Zhejiang University  
Hangzhou, Zhejiang, China  
gmwork@zju.edu.cn

Zhiyao Zhou  
College of Computer Science and  
Technology, Zhejiang University  
Hangzhou, Zhejiang, China  
zjucszy@zju.edu.cn

Jiajun Bu  
Zhejiang Key Lab of Accessible  
Perception and Intelligent Systems,  
Zhejiang University  
Hangzhou, Zhejiang, China  
bjj@zju.edu.cn

## Abstract

Large Language Models (LLMs), empowered by autoregressive next-token prediction, have demonstrated strong reasoning capabilities. Extending this paradigm to graph data requires next graph token prediction, yet existing graph tokens struggle to balance two competing requirements: capturing higher-order structural semantics with high information density, and remaining strictly reversible for faithful decoding. Concretely, first-order textual encodings are reversible but long and semantically sparse, while continuous encodings capture high-level semantics but inevitably lose exact topology. To this end, we propose GraphVulcan, a novel framework that enables discrete reversible and semantic-rich tokenization of graphs using a vocabulary of canonical graphlets. Our approach preserves full structural fidelity while enabling LLMs to natively reason over compositional graphlets through graph-level next-token prediction. We propose a three-stage training paradigm: (1) Structural Semantic Pretraining to learn graph token compositionality, (2) Multi-task Fine-tuning on large-scale CoT-augmented reasoning examples, and (3) Reinforcement Learning to explore and refine reasoning paths. Experiments show that GraphVulcan outperforms first-order encoding baselines across 7 graph reasoning tasks and 3 real-world benchmarks while achieving higher computational efficiency. Our

\*Sheng Zhou is the corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License. *KDD '26, Jeju Island, Republic of Korea*  
© 2026 Copyright held by the owner/author(s).  
ACM ISBN 979-8-4007-2259-2/2026/08  
<https://doi.org/10.1145/3770855.3818177>

work demonstrates that structure-aware discrete tokenization is a feasible way toward general-purpose graph-language models capable of structural reasoning. Codes are available at: [github.com/alibaba-behavioral-risk-control/GraphVulcan](https://github.com/alibaba-behavioral-risk-control/GraphVulcan).

## CCS Concepts

• **Information systems** → **Language models**; • **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → *Reinforcement learning*.

## Keywords

Large Language Models, Graph Vocabulary, Instruction Tuning, Graph Reasoning

### ACM Reference Format:

Zhonghao Wang, Yugang Ji, Zhuonan Zheng, Sheng Zhou, Weigao Wen, Minghao Li, Ming Gu, Zhiyao Zhou, and Jiajun Bu. 2026. Towards Next Graph Token Prediction: Discrete Graph Tokenization for Structural Reasoning in Large Language Models. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '26)*, August 09–13, 2026, Jeju Island, Republic of Korea. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3770855.3818177>

## 1 Introduction

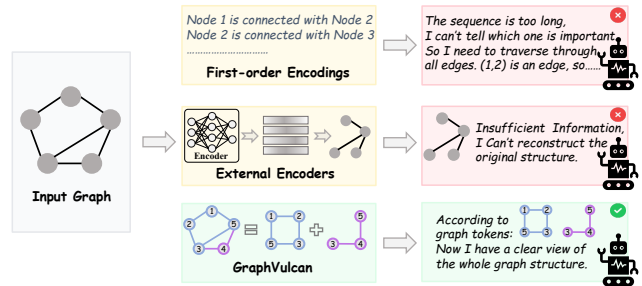
Graph reasoning is the ability to perceive, interpret, analyze and draw conclusions from relational structures, such as determining connectivity, detecting cycles, and computing shortest paths. It is essential across diverse domains, from social networks and biological systems to knowledge graphs and molecular structures.

Yet, conventional graph representation learning methods, including Graph Neural Networks (GNNs) [12, 14, 37] and Graph Transformers (GTs) [4, 40], are ill-suited for graph reasoning because they typically compress a graph into fixed-dimensional continuous embeddings optimized for end-to-end property prediction, obscuring the explicit combinatorial structure needed for step-by-step relational inference. Recent advances in Large Language Models (LLMs) [1] have demonstrated remarkable proficiency in natural language understanding and reasoning [35], and their capabilities have gradually expanded to other modalities such as vision [18] and audio [26]. More recently, these advances have also been extended to graph reasoning, motivating the development of Graph-Language Models (GLMs) [15, 29, 41, 46].

Despite promising progress, most existing GLMs primarily aim at graph understanding or property prediction, and often do not yield an explicit, step-by-step reasoning trace grounded in the underlying topology. Meeting the above requirements calls for an autoregressive *next-token prediction* capability—an objective that is central to LLM reasoning. However, because graphs are discrete and non-sequential, directly aligning graph embeddings with text tokens or naively performing joint training with text does not by itself resolve the challenges discussed above. Specifically, an ideal **Graph Token** should correspond to a canonical structural primitive (e.g., a graphlet [22]) that (i) compactly encodes higher-order topology to maximize information density for Transformer attention and compositional reasoning, and (ii) preserves sufficient structural and node identity information to enable strict reversibility, so that autoregressively predicted token sequences can be deterministically decoded into valid, complete, and faithful graphs.

However, existing graph tokenization approaches often fail to balance representation efficiency and reversibility effectively. First-order textual encodings (e.g., edge lists) [2, 8, 10, 11, 30–32, 43] treat graphs as unstructured bags of edges, offer reversibility but suffer from semantic sparsity, forcing the LLM to expend significant attention bandwidth solely to infer basic high-order motifs like cycles or cliques from atomistic pairwise interactions, thereby crowding out resources for global logical reasoning. Conversely, continuous encodings (e.g., GNN/GT-based projectors or quantized tree vocabularies) [3, 15, 29, 33, 41, 46] capture high-level semantics but incur irreversible topological loss. By compressing discrete combinatorial structures into fixed-width vectors or ambiguous latent codes, they inherently mismatch with the discrete nature of graph data, obscure the precise structural details required for algorithmic reasoning and exact reconstruction. Furthermore, these continuous abstractions often bind the model to identify domain-specific feature without explicit structural reasoning, making it difficult to disentangle whether the model is performing structural reasoning or merely pattern-matching against domain artifacts. Thus, the critical challenge is to construct *discrete* graph tokens that simultaneously (i) preserve exact topology for lossless decoding, and (ii) encapsulate higher-order structural semantics with high information density, so that LLMs can perform *next graph token prediction* and reason compositionally over graph structure.

In this paper, we introduce Graph VocabULary for Compositional SemAntics in Large LaNguage Model, **GraphVulcan**, a novel framework that bridges graph-structured data and LLMs through **reversible, discrete graph tokenization**. To leverage LLMs for



**Figure 1: The motivation behind using graphlet as basic structural primitives. Graphlets capture higher-order topological patterns that go beyond pairwise edges, enabling LLMs to rapidly infer global graph properties and reason compositionally over complex structures.**

graph reasoning, graph must be transform into discrete semantic units that LLMs can predict, compose, and reason over, just as they do with words in natural language. In the graph domain, these units correspond to recurring substructures known as *graphlets* [22], which function as the “words” or “phrases” of a graph-based language and serve as building blocks for complex network topologies. We construct a canonical vocabulary  $\mathcal{V}$  comprising all connected non-isomorphic graphlets, each assigned a unique token. This design achieves exponential compression, significantly improve attention efficiency and semantic density. Moreover, our graphlet vocabulary is fully reversible, effectively preserve local structural semantics thus precisely convey structural information to the LLM.

This vocabulary are then incorporated into the LLM’s vocabulary and deeply integrated with the LLM, enabling next graph token prediction. Based on this graph vocabulary, we design a Structure-Aware Graph Tokenizer that employs a greedy strategy to maximize the semantic information density of each token and leverages Weisfeiler–Lehman (WL) coloring to ensure local-deterministic tokenization. To enable the LLM to understand the compositional relationships and structural hierarchies among different graphlets, we introduce a structure semantic pretraining task. This task trains the LLM to implicitly construct a semantic relational tree over graph tokens within its parameters, thereby capturing the internal structural composition of each token. Building upon this foundation, we further design tailored Chain-of-Thought (CoT) reasoning examples and a reinforcement learning strategy to train the model to perform fundamental graph reasoning tasks. Below are our contributions:

- **Graphlet-based Vocabulary and Tokenization:** We introduce a canonical graph vocabulary comprising all connected non-isomorphic graphlets, where each graphlet is deeply integrate with LLMs to achieve next graph token prediction. Building upon this vocabulary, we propose a structure-aware greedy tokenization algorithm that decomposes arbitrary graphs into compact token sequences with high semantic density, achieving exponential compression while preserving full structural information through reversible encoding.
- **Three-stage Training Curriculum:** We design a hierarchical training strategy including structural semantic pre-training, multi-task fine-tuning, and reinforcement learning

optimization to progressively build the model’s graph reasoning capabilities.

- **Empirical Validation:** Extensive experiments across 7 graph reasoning tasks demonstrate that GraphVulcan significantly outperforms first-order encoding baselines in both accuracy and efficiency.

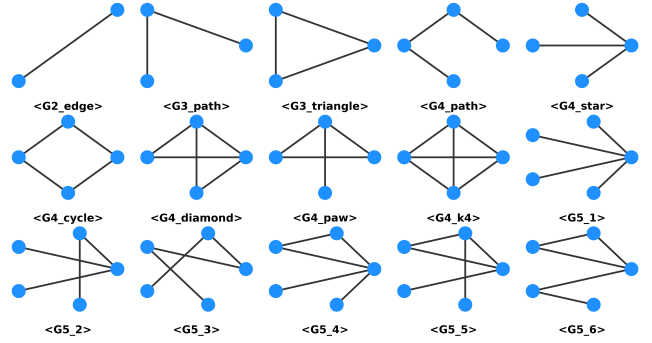
## 2 Related works

### 2.1 Graph Vocabulary

Decomposing graphs into recurring structural primitives—such as network motifs [19] and graphlets [22, 23]—is a well-established approach in network science. These small induced subgraphs serve as universal building blocks that capture local topology across domains (e.g., biological and social networks) and have been widely used for graph comparison [28], classification [16], and anomaly detection [21]. Recent efforts integrate such primitives into representation learning. GFT [33] derives a transferable tree vocabulary from GNN message passing, while GRAVER [39] learns class-specific subgraph patterns via ego-graph disentanglement. However, both encode primitives as continuous embeddings, which are irreversible and lossy [3]. PromptGFM [45] instead uses native LLM tokens to represent nodes as text sequences, avoiding out-of-vocabulary issues in Text-Attributed Graphs—but remains limited to node-level attributes. In contrast, our work tokenizes entire graphs into reversible structural primitives, enabling exact reconstruction, compositional reasoning over global topology, and seamless integration with LLMs through next graph token prediction.

### 2.2 LLM Graph Reasoning

We focus on LLM-based reasoning for fundamental graph algorithmic tasks. Existing approaches fall into two categories: (1) *prompt engineering with frozen LLMs*, and (2) *fine-tuning with graph-aware representations*. For Prompt Engineering. This line of work investigated how different prompting strategies can elicit graph understanding from serialized inputs. Works like NLGraph [31] evaluate how natural language descriptions of graphs, combined with strategies such as Build-a-Graph and Algorithmic Prompting, elicit reasoning in LLMs. GraphQA [8] systematically studies the impact of different graph serialization methods on LLM performance. For Instruction Tuning this line of work adapts LLMs via fine-tuning. GraphWiz [2] uses a large instruction-tuning dataset with explicit reasoning traces and applies DPO [24]. NLGift [43] examines generalization across graph patterns, while InstructGraph [32] enhances reasoning and generation via instruction tuning and a code-like graph verbalizer—avoiding external encoders. However, these methods typically rely on first-order encodings (e.g., edge or adjacency lists), which capture only pairwise relations and miss higher-order motifs. Some incorporate multi-hop neighborhoods [3, 44], but often sacrifice global connectivity and lack reversibility. Others deploy LLM agents with external tools [17, 42], yet these focus on task execution rather than fostering intrinsic graph understanding. To enable structural reasoning in LLMs, we argue that graphs should be represented using fundamental structural primitives—such as graphlets—that support interpretable, semantic-rich, and reversible tokenization within LLMs.



**Figure 2: First 15 non-isomorphic graphlets with node count  $2 \leq k \leq 5$  in graph vocabulary<sup>1</sup>.**

## 3 Method

### 3.1 Preliminary

In this section, we introduce the fundamental concepts underlying our graph tokenization approach. We first define the graph tokenization problem and then discuss the notion of structural semantics that guides our vocabulary design.

**3.1.1 Graph Tokenization.** The tokenization process decomposes the input graph  $G$  into a sequence of structural units (tokens), each representing a subgraph pattern from the vocabulary. This transformation enables graph structures to be processed by LLMs.

*Definition 3.1 (Graph Tokenization).* Given an undirected graph  $G = (V, E)$ , **graph tokenization** is a mapping function  $f : \mathcal{G} \rightarrow \mathcal{T}^*$  that transforms the graph into a finite sequence of tokens from a predefined vocabulary  $\mathcal{V}$ :

$$f(G) = T = [t_1, t_2, \dots, t_n], \quad t_i \in \mathcal{V}. \quad (1)$$

To guarantee full reversibility of the token sequence, it must encode not only structural information but also node-specific information, ensuring a complete and unambiguous reconstruction of the original graph. Under this definition, various first-order encoding approaches (e.g., Edge list [31], Incident [8]) can be considered as forms of graph tokenization with vocabulary  $\mathcal{V}$  that only consists of nodes and edges. However, an ideal graph token sequence should also exhibit high information density—i.e., each token should carry as much structural information as possible to enhance the efficiency of the attention mechanism in Transformer-based models. Under this objective, first order encoding is suboptimal due to its low information density.

### 3.2 Graphlet-based Vocabulary

The tokenization of graph structures in language models presents a fundamental challenge: how to encode complex topological information in a way that preserves structural semantics while remaining computationally tractable. To better illustrate the limitations of existing graph encodings, let us consider a specific scenario. As shown in Figure 1, when graph structures are first-order encoded

<sup>1</sup>For full vocabulary please refer to Appendix B

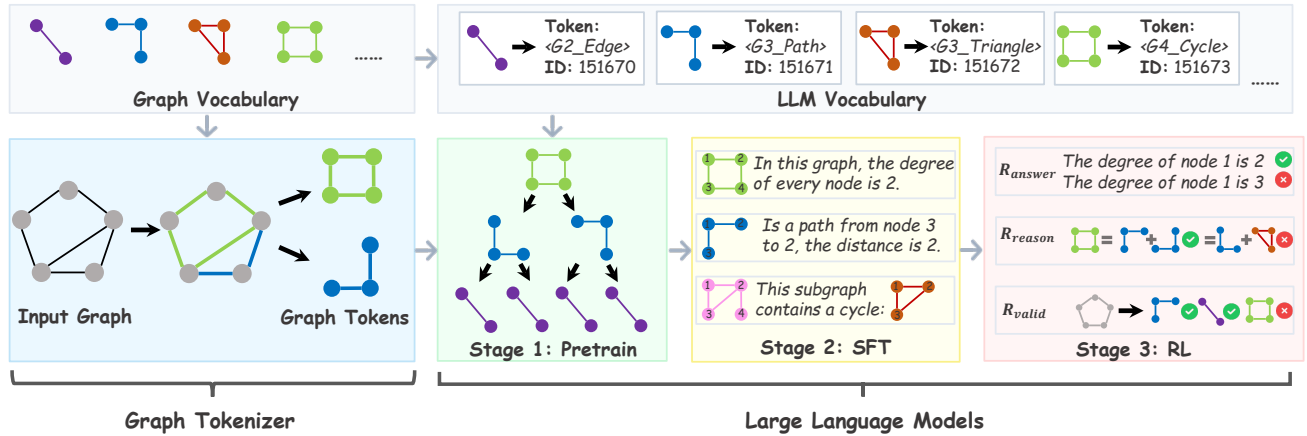


Figure 3: Training pipelines of GraphVulcan.

as natural-language text, such as edge lists, the resulting tokenization can preserve most raw structural information and is therefore largely reversible. However, the encoded token sequence is often excessively long and semantically sparse: the model must implicitly reconstruct higher-order structures from scattered pairwise edges and cannot directly identify which motifs or substructures are relevant to the task. External graph encoders provide an alternative by compressing high-order structural information into continuous dense vectors, but this compression is inherently lossy, making it difficult or impossible for the LLM to recover the original topology from the dense embedding vectors. In contrast, graphlet token sequences preserve reversibility while directly encoding higher-order structural semantics, enabling the LLM to access task-relevant graph structures without reconstructing them from raw edges.

In this work, we propose a graphlet-based vocabulary that decomposes graphs into a finite set of small, connected subgraph patterns. This design is motivated by two key principles: (1) **Information Density**: graphlets achieve exponential compression. A  $k$ -clique contains  $k(k-1)/2$  edges but requires only a single graphlet token, yielding a compression ratio of  $O(k^2) \rightarrow O(1)$ . This dramatically reduces sequence length from  $O(|E|)$  to  $O(|E|/k)$ , directly improving Transformer attention efficiency and enabling the model to capture global graph properties within limited context windows. (2) **Semantic Preservation**: Our graphlet vocabulary is fully reversible by preserving all structural information. Moreover unlike edge-by-edge representations, graphlets preserve local topological context. For example, a triangle (3-clique) conveys fundamentally different structural information than a 3-path, and this distinction is explicitly encoded in our vocabulary. This design aligns with human cognitive patterns: when reasoning about networks, humans naturally identify motifs (e.g., "a social clique", "a hub-and-spoke structure") rather than enumerating individual edges.

Figure 2 illustrates a subset of the graph structure vocabulary, for the full structure vocabulary please refer to Appendix B. These fundamental structural units are already capable of effectively capturing local structural invariants. However, in addition to these

structural tokens, we also need to define a set of special tokens to form a complete representation of the graph:

$\langle G1 \rangle$ : Denotes isolated nodes in the graph.

$\langle \text{NidB} \rangle / \langle \text{NidS} \rangle / \langle \text{NidE} \rangle$ : Mark which nodes are included in each structural token, where  $\langle \text{NidB} \rangle / \langle \text{NidS} \rangle$  marks the start/end of node ID sequence and  $\langle \text{NidS} \rangle$  splits them; without this information, the completeness objective cannot be achieved.

$\langle G\_Operator\_Eq \rangle$ : Functions similarly to an equals sign, indicating semantic equivalence between the token sequence on its left and right sides; this operator plays a crucial role during inference.

$\langle G\_Connect \rangle / \langle G\_Disconnect \rangle$ : Used to annotate each connected component in the graph. We consider connectivity a key aspect of structural semantics—since different connected components are semantically independent, they require explicit marking.

Through the above definitions, we obtain a full graph vocabulary  $\mathcal{V}$ . Notably, we use distinct mnemonics to denote graph tokens; but this does not imply that the LLM can directly access the natural language tokens corresponding to these mnemonics. These tokens are actually added to the LLM’s vocabulary as atomic units, and the dimensions of the LLM’s embedding and projection head are adjusted accordingly to accommodate the new vocabulary. This enables the LLM not only to understand these graph tokens but also to generate them as output, thereby facilitating next graph token prediction.

### 3.3 Structure-Aware Graph Tokenization

With the graph vocabulary  $\mathcal{V}$  defined, the core challenge of graph tokenization is to transform an arbitrary graph  $G$  into a linear sequence of tokens  $T$  that is both *compact* and *reversible*. To achieve these dual goals, we propose a **Structure-Aware Graph Tokenization** algorithm, the overall procedure is depicted in Algorithm 1.

**3.3.1 Algorithm Overview.** Our approach iteratively decomposes the graph by extracting subgraphs that maximize local edge density, guided by structural invariants to resolve ambiguities.

The tokenization proceeds in three phases:

1. *Structural Initialization.* For every node  $v$  in the graph  $G$ , we precompute its structural features to ensure deterministic tokenization. We compute two key metrics:

**Weisfeiler-Lehman Hash:** To ensure consistent tokenization across isomorphic graphs, we employ the Weisfeiler-Lehman (WL) [36] algorithm hash to assign unique signature. For each node  $v \in V$ , we compute its structural signature  $h(v)$  through iterative hash updates:

$$\begin{aligned} h^{(0)}(v) &= \text{degree}(v), \\ h^{(t+1)}(v) &= \text{hash}\left(h^{(t)}(v), \{\{h^{(t)}(u) : u \in \mathcal{N}(v)\}\}\right), \end{aligned} \quad (2)$$

where  $\mathcal{N}(v)$  denotes the neighborhood of node  $v$ , and  $\{\{\cdot\}\}$  represents a multiset. After  $T$  iterations, we obtain the final WL-hash  $h(v) = h^{(T)}(v)$  for each node. In this work, we typically perform 5 rounds of WL iterations as a conservative measure. These WL hashes are subsequently used for graphlet matching and node ordering. We adopt 1-WL mainly as a trade-off between reversibility and computational efficiency. Notably, although in our graph vocabulary 1-WL is enough for distinguishing all graphlets  $k \leq 5$ . If  $k > 5$ , 1-WL may produce identical hashes for structurally distinct graphs, which would compromise the reversibility. In such cases, we recommend employ more advanced techniques, such as k-WL.

**Clustering Coefficient:** To prioritize nodes with high local connectivity, we compute the clustering coefficient [34] for each node  $v$ :

$$C(v) = \frac{2 \cdot |\{(u, w) \in E : u, w \in \mathcal{N}(v)\}|}{|\mathcal{N}(v)| \cdot (|\mathcal{N}(v)| - 1)}. \quad (4)$$

It measures the fraction of possible triangles through node  $v$  that actually exist in the graph, capturing global subgraph density.

These features remain static or are efficiently updated to ensure that node selection depends solely on graph topology rather than arbitrary IDs.

2. *Deterministic Subgraph Extraction.* As depicted in Algorithm 2, the graph is decomposed via an iterative "extract-and-remove" procedure. We initialize the residual graph  $G_{res} = G$ . In each step, we identify a subgraph  $S$  from  $G_{res}$  using a greedy strategy refined by structural signals:

**Start Node Selection:** To initiate a new token, we select a node  $v_{start}$  that acts as a structural "anchor." To achieve compactness and determinism, we prioritize nodes with high structural density, ranking them by the tuple  $(C(v), h(v))$  in descending order. This ensures that the decomposition consistently begins at the most distinctive regions of the graph (e.g., the center of a star or a node in a clique).

**Greedy Expansion:** We grow the subgraph  $S$  by iteratively adding neighbors from  $\mathcal{N}(v)$ . For each candidate node  $u \notin S$ , we define the *local subgraph density score*:

$$S_{\text{density}}(u) = \sum_{v \in S} \mathbb{I}[(u, v) \in E], \quad (5)$$

where  $\mathbb{I}[\cdot]$  is the indicator function. This score counts the number of edges connecting  $u$  to the current subgraph  $S$ . This greedy heuristic naturally favors dense motifs (e.g., triangles, K4) that encapsulate more edges per token than paths.

**Ranking** Crucially, candidates are ranked by the tuple:

$$\text{Rank}(u) = (S_{\text{edge}}(u), C(u), h(u)) \quad (6)$$

in descending lexicographic order. When multiple candidates offer the same score, we break ties using their clustering coefficient and WL signatures. This step guarantees determinism: identical structural contexts will always be traversed in the same order, making the tokenization invariant.

3. *Sequence Serialization.* Once a subgraph  $S$  is extracted: 1) We map  $S$  to a graph token  $t \in \mathcal{V}$  via its canonical WL-hash. 2) The token is then concatenated with Node ID information as  $\langle \text{NidB} \rangle v_1 \text{NidS} \dots v_k \langle \text{NidE} \rangle t$ . To ensure completeness and reversibility, we use the VF2 [5] algorithm to obtain a mapping  $VF2 : S \rightarrow t$  from each extracted subgraph to its corresponding token  $t \in \mathcal{V}$ , and then sort the node IDs according to the order of their original IDs within the mapped token  $t$ . This practice can ensure a reversible canonical internal order. 3) The edges of  $S$  are removed from  $G_{res}$ . If the extraction fragments a connected component, we insert a  $\langle \text{G\_Connect} \rangle$  token to explicitly preserve the global connectivity information between the extracted subgraph and the remaining graph.

These processes repeats until  $G_{res}$  is empty. Disconnected components are separated by  $\langle \text{G\_Disconnect} \rangle$  tokens, ensuring the final sequence can fully reconstructs the topology of  $G$ .

## 3.4 Training Strategy

To enable LLMs to effectively reason over graph structures using our graphlet vocabulary, we propose a three-stage hierarchical training curriculum that progressively builds the model's graph understanding capabilities.

3.4.1 *Stage 1: Structural Semantic Pretraining.* The newly added Graph Tokens, in fact, carry no semantic information initially, so a pretraining process is required to endow them with basic semantics. The first stage focuses on teaching the LLM fundamental operations over graph tokens through decomposition and merge tasks. The core objective is to enable the model to internalize the compositional relationships among graphlets, implicitly constructing a **semantic tree** of graph tokens within its parameters.

*Task Design.* We generate training data by systematically enumerating all possible decompositions of graphlets in our vocabulary. For each graphlet token  $t$  with  $k$  nodes and  $m$  edges, we identify all valid binary partitions  $(t_1, t_2)$  such that:

- Both  $t_1$  and  $t_2$  are connected subgraphs
- $t_1 \cup t_2 = t$  (edge-disjoint union)
- Both  $t_1$  and  $t_2$  exist in the vocabulary  $\mathcal{V}$

For each valid partition, we create two types of training samples:

- **Decompose:** Given  $t$ , predict the equivalent sequence  $t_1 + t_2$
- **Merge:** Given  $t_1 + t_2$ , predict the equivalent token  $t$

*Training Objective.* The model learns to recognize structural equivalence through the operator  $\langle \text{G\_Operator\_Eq} \rangle$ :

$$t = t_1 + t_2 \implies t \langle \text{G\_Operator\_Eq} \rangle t_1 \langle \text{G\_Connect} \rangle t_2 \quad (7)$$

$$t_1 + t_2 = t \implies t_1 \langle \text{G\_Connect} \rangle t_2 \langle \text{G\_Operator\_Eq} \rangle t \quad (8)$$

This pretraining phase establishes the foundational understanding that complex graph structures can be decomposed into simpler primitives and vice versa, enabling compositional reasoning in later stages.

**3.4.2 Stage 2: Graph Reasoning Supervised Fine-tuning.** Building upon the structural semantic foundation, the second stage trains the model on diverse graph reasoning tasks with carefully designed chain-of-thought (CoT) reasoning paths.

*Task Coverage.* We construct training data for seven fundamental graph reasoning tasks spanning different computational complexity levels. The detailed information are shown in Tabel 1, we will introduce them in the experiment section.

*CoT Design Philosophy.*<sup>2</sup> The core idea is to train the model to identify and reason over **key tokens** relevant to tasks through reasoning templates. We illustrate this with the cycle detection task as an example. Given a graph token sequence, the CoT generation process identifies all graphlet tokens that form a closed walk (cycle). For each edge in the detected cycle, we locate the corresponding graphlet token covering that edge, then construct a token sequence representing the cycle by connecting these tokens with  $\langle G\_Connect \rangle$  separators. The reasoning explicitly demonstrates the structural equivalence: the original graph can be decomposed into the cycle subgraph and residual components, formally expressed as  $G \langle G\_Operator\_Eq \rangle G_{cycle} \langle G\_Connect \rangle G_{residual}$ , where  $G_{cycle}$  represents the identified cyclic structure. This structured reasoning process teaches the model to parse task descriptions to identify relevant structural properties and apply task-specific reasoning strategies by tracing token sequences, thereby generate explicit, step-by-step reasoning chains that mirror algorithmic thinking.

**3.4.3 Stage 3: Reinforcement Learning for Reasoning Path Exploration.** The third stage addresses a critical limitation of supervised fine-tuning: the reasoning paths learned in Stage 2 tend to be overly deterministic and may not generalize to diverse problem-solving strategies. We employ Group Relative Policy Optimization (GRPO) [27] to encourage the model to explore alternative valid reasoning paths.

*Multi-Component Reward Function.* To ensure both correctness and reasoning validity, we design a composite reward function with three components:

**1. Answer Reward ( $R_{answer}$ ):** Binary reward based on whether the final answer matches the ground truth:

$$R_{answer} = \begin{cases} \alpha & \text{if answer is correct} \\ 0.0 & \text{otherwise} \end{cases}, \quad (9)$$

where we set scaling factor  $\alpha = 1.4$  in this paper.

**2. Hallucination Penalty ( $R_{valid}$ ):** To suppress hallucinated graph tokens, we verify that all graphlet tokens used in the reasoning process exist in the original input graph. For each graphlet token  $g_i$  in the model’s output, we check if all its edges are present in the input graph  $G$ :

$$R_{valid} = \beta \cdot \frac{\sum_{i=1}^n \mathbb{I}[\text{edges}(g_i) \subseteq E(G)]}{n}, \quad (10)$$

where  $n$  is the number of graphlet tokens (excluding single-node tokens) in the reasoning chain, and we set scaling factor  $\beta = 1.0$ . This reward component ranges from 0.0 to  $\beta$  and penalizes the model for inventing non-existent graph structures.

<sup>2</sup>For training data examples, please refer to Appendix D

**3. Reasoning Reward ( $R_{reasoning}$ ):** To prevent the model from reaching correct answers through invalid reasoning paths, we validate the structural correctness of graph operations. Specifically, we check whether the graphlet tokens used in reasoning form valid subgraphs and whether the reasoning steps follow legal graph operations (e.g., proper use of  $\langle G\_Connect \rangle$  and  $\langle G\_Disconnect \rangle$  tokens, and the validity of the structural equivalence expressions). This component ranges from 0.0 to  $\gamma$ , and we set scaling factor  $\gamma = 0.6$ .

*Final Reward.* The total reward is the sum of all three components:

$$R_{total} = R_{answer} + R_{valid} + R_{reasoning}. \quad (11)$$

This multi-faceted reward design ensures that the model not only produces correct answers but also maintains structural validity and reasoning coherence throughout the inference process.

*Training Dynamics.* We employ **curriculum learning** by gradually increasing task difficulty to stabilize the GRPO training process. To prevent potential policy collapse, we select the checkpoint with the highest reward on the training data as the final model. This stage significantly improves the model’s ability to discover diverse, valid reasoning strategies while maintaining high accuracy.

## 4 Experiment

In this section, we conduct comprehensive experiments to analyze the graph reasoning performance of GraphVulcan.

### 4.1 Task Descriptions

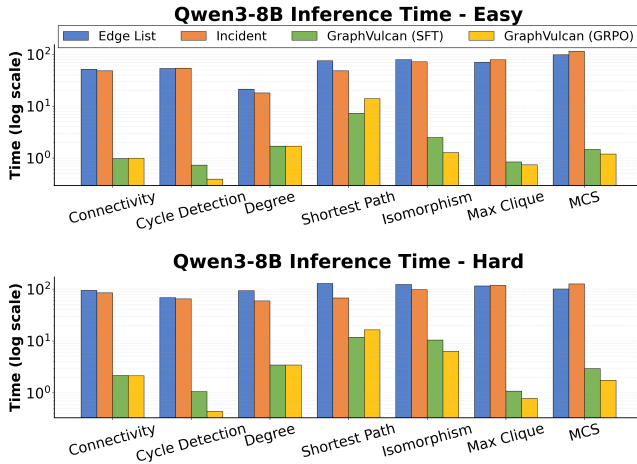
We evaluate GraphVulcan on seven fundamental graph reasoning tasks spanning different computational complexity: Polynomial-time(P), Nondeterministic Polynomial-time(NP) and Nondeterministic Polynomial-time Complete(NP-C). **Connectivity (P)** Given a graph  $G = (V, E)$  and two nodes  $u, v \in V$ , determine whether there exists a path connecting  $u$  and  $v$ . **Cycle Detection (P)** Given a graph  $G = (V, E)$ , determine whether  $G$  contains at least one cycle. **Degree Calculation (P)** Given a graph  $G = (V, E)$  and a target node  $v \in V$ , compute the degree of  $v$ . **Shortest Path (P)** Given a graph  $G = (V, E)$  and two nodes  $u, v \in V$ , find the length of the shortest path between  $u$  and  $v$ . **Graph Isomorphism (NP)** Given two graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ , determine whether there exists a bijection preserving edge relationships. **Maximum Clique (NP-C)** Given a graph  $G = (V, E)$ , find the largest complete subgraph where every pair of nodes is connected. **Maximum Common Subgraph (NP-C)** Given two graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ , find the largest common isomorphic subgraphs.

### 4.2 Experimental Setup

*Training Data Configuration.* We evaluate GraphVulcan on seven fundamental graph reasoning tasks spanning different computational complexity classes. Table 1 summarizes the training data configuration for each task, including graph size ranges, dataset sizes for both SFT and GRPO, random graph generation models, and test data specifications. For each task, we generate training graphs using random graph models with carefully chosen parameters to ensure diverse structural patterns. The Erdos-Renyi [7] model with  $p = 0.3$  provides a balanced edge density suitable for most tasks, while the  $n$ - $m$  random graph [7] model with  $m = n - 1$  for cycle

**Table 1: Training Data Configuration for Seven Graph Reasoning Tasks**

Task	Complexity	Training Nodes	SFT Samples	GRPO Samples	Random Graph Model	Graph Parameters	Test Nodes (Easy)	Test Nodes (Hard)
Connectivity	Polynomial	11-50	20,000	10,000	Erdos-Renyi	$p = 0.3$	11-30	31-50
Cycle Detection	Polynomial	11-50	20,000	10,000	$n$ - $m$ Random	$m = n - 1$	11-30	31-50
Degree	Polynomial	11-50	26,200	10,000	Erdos-Renyi	$p = 0.3$	11-30	31-50
Shortest Path	Polynomial	11-50	20,000	10,000	Erdos-Renyi	$p = 0.3$	11-30	31-50
Isomorphism	NP	6-12	20,000	10,000	Erdos-Renyi	$p = 0.3$	6-10	11-15
Max Clique	NP-Complete	5-10	20,000	10,000	Erdos-Renyi	$p = 0.3$	5-7	8-10
Max Common Subgraph	NP-Complete	5-10	20,000	10,000	Erdos-Renyi	$p = 0.3$	5-7	8-10

**Figure 4: Inference time (per sample) comparison of GraphVulcan against other encoding methods, using Qwen3-8B as the base model, with the y-axis shown on a log scale.**

detection ensures cyclic and acyclic graphs have same number of edges. Training node ranges are selected based on task complexity: polynomial-time tasks (connectivity, cycle detection, degree, shortest path) use larger graphs (11-50 nodes), while NP-complete tasks (max clique, max common subgraph) and NP tasks (isomorphism) use smaller graphs (5-10 and 6-15 nodes respectively) due to higher difficulty. Test data is split into "Easy" and "Hard" categories based on graph size to evaluate model performance across different difficulty levels. All test sets contain 300 samples, split into 3 subsets of 100 each; we report the mean and standard deviation over 3 runs.

**Baseline Models and Encoding Methods.** We compare GraphVulcan against three open-source LLMs: **Qwen3-8B** [38], **LLaMa3.1-8B** [6], and **Deepseek-R1-Distill-Qwen-7B** [9], as well as the advanced closed-source model **GPT-4o** [13], in which Qwen3-8B serves as the base model of GraphVulcan. These models represent diverse architectural designs and training paradigms, providing a comprehensive evaluation landscape. For graph encoding, in addition to our GraphVulcan’s graph vocabulary encoding, we employ two first-order representations as baselines: **Edge List** encoding, which serializes graphs as sequences of node pairs  $(u, v)$  representing edges, this encoding method is widely used in previous studies [8, 31, 43], and **Incident** encoding, which represents each

node followed by its adjacent neighbors in the format "Node 1 is connected with Node 2, 3, 4", it achieves best overall performance as reported by GraphQA [8] benchmark. All baseline models are evaluated using these two encoding methods to ensure fair comparison with GraphVulcan’s GraphVocab encoding. We run each experiment 3 times and record the mean accuracy correspond with its standard deviation.

**Hyperparameters and Hardware.** All trainings are conducted using DeepSpeed ZeRO-2 and ZeRO-3 [25] for distributed training optimization. For the Stage 1, we train for 3 epochs with a learning rate of  $5 \times 10^{-5}$ , For the Stage 2, we train for 1 epoch with a learning rate of  $5 \times 10^{-5}$ . Both stages employ a cosine learning rate scheduler with 3% warmup steps. For the GRPO stage, we use a learning rate of  $5 \times 10^{-6}$  and train for 1 epoch, with a KL penalty coefficient of 0.03 to prevent the policy from deviating too far from the reference model. All training is performed on 8×NVIDIA H20-141GB GPUs with bfloat16 precision. We use 1×NVIDIA L20 GPU for inference, and we limit maximum generation length of 8192 tokens. The temperature is set to 0.7. Batch size is set to 10.

### 4.3 Main Results

Table 2 presents comprehensive experimental results across seven graph reasoning tasks, comparing GraphVulcan against baseline LLMs using first-order encodings (Edge List and Incident). The results reveal several key findings:

**Consistent superiority across most tasks.** GraphVulcan outperforms all baseline methods across most tasks. For polynomial-time tasks, GraphVulcan achieves near-perfect accuracy on most test cases. Specifically, on Connectivity (Easy), GraphVulcan-GRPO achieves 100.00% accuracy compared to the best baseline (Qwen3-8B with Incident encoding) at 98.33%. On Cycle Detection, both GraphVulcan-SFT and GraphVulcan-GRPO achieve 100.00% accuracy on Easy cases, while the best baseline reaches only 93.67%. However, we also observed that on the shortest path length task, while GraphVulcan-GRPO achieving the second-best performance and outperforming all EdgeList-based encoding baselines—still fell short of the Qwen3-8B + Incident encoding. We hypothesize this is due to the intrinsic nature of the shortest path task: it requires exhaustive traversal of all edges to enumerate all possible paths and identify the shortest one. In contrast, GraphVulcan’s training data encourages the model to reach conclusions in as few reasoning steps as possible—a strategy that reduces inference time but

**Table 2: Experimental Results on 7 Graph Reasoning Tasks (3 Runs).**

Task	Model Difficulty	LLaMa3.1-8B		Deepseek-R1-Distill-Qwen-7B		GPT-4o		Qwen3-8B			
		Edge List	Incident	Edge List	Incident	Edge List	Incident	Edge List	Incident	GraphVulcan (SFT)	GraphVulcan (GRPO)
Connectivity	Easy	58.33 ± 4.93%	63.00 ± 2.00%	69.33 ± 4.16%	86.00 ± 3.00%	95.00 ± 1.00%	97.00 ± 1.00%	95.33 ± 2.31%	98.33 ± 1.15%	99.67 ± 0.57%	100.00 ± 0.00%
	Hard	53.67 ± 0.57%	58.33 ± 1.53%	60.33 ± 4.73%	72.33 ± 1.53%	88.67 ± 3.79%	94.00 ± 1.00%	73.00 ± 1.73%	86.33 ± 3.79%	100.00 ± 0.00%	100.00 ± 0.00%
Cycle Detection	Easy	57.00 ± 3.61%	51.67 ± 6.43%	66.33 ± 2.52%	80.33 ± 6.66%	67.33 ± 3.51%	68.67 ± 4.04%	93.00 ± 3.00%	93.67 ± 1.53%	100.00 ± 0.00%	100.00 ± 0.00%
	Hard	53.33 ± 7.37%	55.67 ± 2.08%	56.67 ± 6.43%	73.67 ± 3.51%	56.00 ± 2.00%	58.67 ± 2.08%	82.67 ± 1.53%	93.00 ± 1.73%	99.67 ± 0.57%	100.00 ± 0.00%
Degree	Easy	42.33 ± 2.52%	57.33 ± 4.04%	60.33 ± 6.81%	71.67 ± 7.57%	67.00 ± 5.29%	91.33 ± 3.79%	94.00 ± 2.65%	98.67 ± 1.53%	99.67 ± 0.57%	100.00 ± 0.00%
	Hard	7.67 ± 0.57%	23.33 ± 6.43%	13.33 ± 4.16%	59.67 ± 2.52%	31.00 ± 3.46%	82.67 ± 4.16%	55.33 ± 4.51%	94.33 ± 1.53%	97.33 ± 2.08%	91.67 ± 4.04%
Shortest Path	Easy	21.67 ± 2.08%	40.67 ± 4.73%	59.67 ± 6.43%	85.00 ± 5.20%	57.00 ± 6.56%	62.33 ± 2.31%	88.00 ± 3.00%	100.00 ± 0.00%	86.00 ± 1.73%	93.00 ± 2.65%
	Hard	8.67 ± 4.16%	34.00 ± 4.58%	57.67 ± 6.43%	82.00 ± 3.46%	33.00 ± 2.00%	44.00 ± 5.57%	80.00 ± 3.61%	98.33 ± 0.57%	73.67 ± 4.93%	89.67 ± 4.04%
Isomorphism	Easy	49.44 ± 1.52%	48.48 ± 3.62%	53.00 ± 1.73%	67.00 ± 3.61%	70.49 ± 0.95%	70.84 ± 3.16%	67.67 ± 2.52%	84.33 ± 0.57%	98.67 ± 0.57%	89.00 ± 5.57%
	Hard	52.67 ± 1.88%	49.89 ± 3.94%	51.00 ± 1.00%	58.67 ± 2.31%	54.54 ± 2.17%	61.03 ± 4.60%	54.33 ± 1.53%	81.33 ± 2.08%	97.33 ± 1.15%	85.33 ± 3.05%
Max Clique	Easy	9.67 ± 1.15%	10.33 ± 1.53%	64.67 ± 3.79%	65.67 ± 5.51%	75.33 ± 8.96%	67.00 ± 3.00%	91.67 ± 0.57%	92.33 ± 1.53%	99.67 ± 0.57%	99.33 ± 0.57%
	Hard	3.00 ± 2.00%	5.67 ± 2.52%	30.33 ± 4.51%	38.67 ± 6.11%	52.33 ± 7.77%	51.67 ± 3.05%	71.00 ± 5.00%	65.67 ± 3.21%	95.67 ± 3.21%	98.67 ± 0.57%
Max Common Subgraph	Easy	0.33 ± 0.57%	0.33 ± 0.57%	11.33 ± 6.35%	16.67 ± 2.31%	2.33 ± 1.53%	8.67 ± 2.08%	12.00 ± 3.61%	22.00 ± 1.73%	61.00 ± 3.46%	63.33 ± 2.08%
	Hard	0.00 ± 0.00%	0.00 ± 0.00%	0.67 ± 0.57%	0.67 ± 0.57%	0.00 ± 0.00%	0.33 ± 0.57%	0.00 ± 0.00%	1.00 ± 1.73%	6.00 ± 3.00%	4.33 ± 4.04%
<b>Average</b>		29.84%	35.62%	46.76%	61.29%	53.57%	61.30%	68.43%	79.24%	86.74%	86.74%

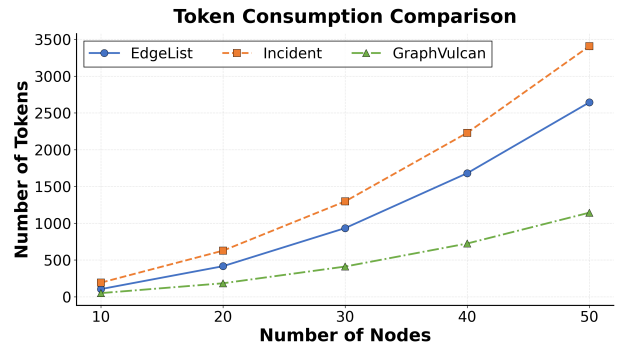
inherently limits exhaustive enumeration, potentially leading to suboptimal solutions. We will address this limitation in future work.

**Improvements on complex tasks.** The performance gap becomes even more pronounced on NP and NP-Complete tasks. For Graph Isomorphism (Hard), GraphVulcan-SFT achieves 97.33% accuracy, significantly outperforming the best baseline (Qwen3-8B with Incident) at 81.33%. On Maximum Clique (Hard), GraphVulcan-GRPO reaches 98.67% accuracy compared to the best baseline’s 71.00%. Most strikingly, on Maximum Common Subgraph (Easy), GraphVulcan-GRPO achieves 63.33% accuracy while the best baseline only achieves 22.00%, on Maximum Common Subgraph (Hard), GraphVulcan-SFT achieves a 6.00% accuracy while all baseline methods fail completely ( $\leq 1.00\%$ ), demonstrating that GraphVulcan encoding achieve higher structural expressiveness.

**SFT training is enough for most cases.** We compare GraphVulcan-SFT and GraphVulcan-GRPO in Table 2. For most tasks (Connectivity, Cycle Detection, Degree, Max Clique, Max Common Subgraph), both SFT and GRPO variants achieve comparable accuracy, suggesting that supervised fine-tuning alone is sufficient for most graph reasoning tasks. The structured nature of graphlet tokens and the clear reasoning paths in CoT training data enable the model to learn effective strategies through supervised learning. We also noticed that GRPO demonstrates clear advantage on Shortest Path task, GRPO achieves 89.67% compared to SFT’s 73.67%. On Isomorphism, GRPO show lower accuracy on compared with SFT. These results suggest that while reinforcement learning helps the model discover diverse reasoning strategies in some cases, GraphVulcan-SFT already provides a strong foundation for graph reasoning.

## 5 Computational Efficiency

Figure 4 presents inference time per sample across all tasks, we use identical inference device and parameter to ensure fair comparison. GraphVulcan demonstrates exponential computational efficiency compared to baseline methods, with inference times typically 5-50× faster. For Connectivity (Easy), GraphVulcan requires only 0.98 seconds per sample compared to 48.22 seconds for the best baseline (Qwen3-8B with Incident), representing a 49× speedup. On Cycle



**Figure 5: Token consumption comparison across different graph encoding methods. For each data point, we report the average token consumption over 10 randomly generated Erdős-Rényi graphs with  $p=0.3$ .**

Detection (Hard), GraphVulcan is 61× faster than Incident. Interestingly, GraphVulcan maintains efficiency even on complex tasks. On Maximum Common Subgraph (Hard), GraphVulcan-SFT requires only 2.94 seconds compared to 125.81 seconds for the baseline, which is a 43× speedup. This dramatic efficiency gain stems from two key factors: (1) **Training strategy:** The CoT reasoning path in stage 2 training data encourages the model to quickly identify, decompose, or merge task-relevant key tokens and directly generate answers based on them. (2) **Semantic density:** Each graphlet token encodes rich structural information, enabling the model to perform reasoning with fewer operations.

## 6 Token Usage Comparison

To analyze the extent to which GraphVulcan can enhance the information density of graph tokens, we conduct a controlled experiment comparing token consumption on across three encoding methods: EdgeList, Incident, and GraphVocab. We generate random Erdős-Rényi graphs with varying node counts with edge probability  $p = 0.3$ , then measure the actual token count after encoding with

**Table 3: Graph classification accuracy (%) on real-world TU-Dataset benchmarks. All methods use Qwen3-8B with identical SFT training. Results are 5-fold cross-validation (mean  $\pm$  std). Best results in bold.**

Dataset	Edge List	Incident	GraphVulcan
IMDB-BINARY	48.40 $\pm$ 2.89	49.90 $\pm$ 1.77	<b>59.70 <math>\pm</math> 2.93</b>
IMDB-MULTI	36.80 $\pm$ 1.20	34.73 $\pm$ 1.62	<b>40.80 <math>\pm</math> 2.58</b>
REDDIT-BINARY	52.55 $\pm$ 1.53	49.55 $\pm$ 1.96	<b>65.15 <math>\pm</math> 3.36</b>
<b>Average</b>	45.92	44.73	<b>55.22</b>

the Qwen3-8B tokenizer. For each node count, we average results over 10 randomly generated graphs to ensure statistical reliability.

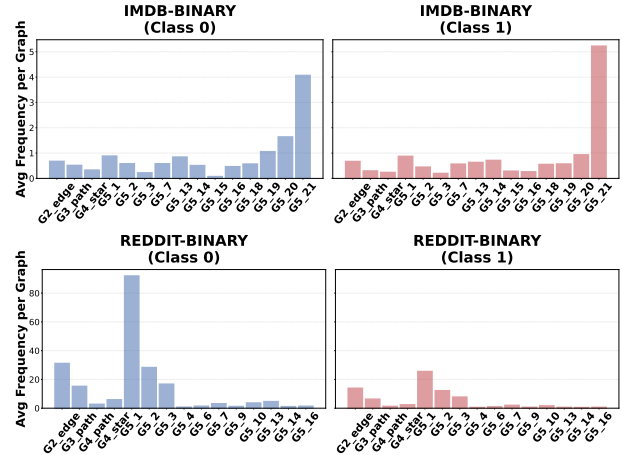
Figure 5 presents the token consumption comparison. The results demonstrate that GraphVocab achieves a stable compression ratio of approximately  $2\times$  compared to EdgeList across all graph sizes. Specifically, for 10-node graphs, GraphVocab requires only 47.1 tokens versus EdgeList’s 102.5 tokens. This advantage scales consistently: at 50 nodes, GraphVocab uses 1138.3 tokens compared to EdgeList’s 2650.1 tokens, maintaining the  $2.3\times$  compression ratio. The compression stems from the greedy tokenization strategy and graphlets’ ability to encode  $O(k^2)$  edges in a single token. A 5-clique containing 10 edges is represented by one  $\langle G5\_21 \rangle$  token sequence, while EdgeList requires 10 separate edge token sequences. This information density directly translates to reduced sequence length, which quadratically improves Transformer attention efficiency.

## 7 Preliminary Results on Real-World Graphs

To validate whether graphlet-based tokenization generalizes beyond synthetic graphs, we conduct preliminary experiments on real-world graph classification benchmarks from the TUDataset [20]. We select three social network datasets: IMDB-BINARY, IMDB-MULTI, and REDDIT-BINARY, whose average node counts are 19.77, 13.00, and 429.63, respectively. These datasets exhibit diverse structural characteristics representative of real-world graph distributions.

We also compare GraphVulcan against two first-order encoding baselines Edge List and Incident. For the baselines, we use Qwen3-8B as the base model; for GraphVulcan, we use GraphVulcan-GRPO. All models are trained with identical SFT procedures and hyperparameters. We perform standard 5-fold cross-validation and report mean accuracy with standard deviation. As shown in Table 3, GraphVulcan consistently outperforms both first-order encoding baselines across all datasets, achieving an average accuracy of 55.22%, surpassing Edge List by +9.30% and Incident by +10.49%. The improvement is pronounced on larger dataset REDDIT-BINARY, whose GraphVulcan achieves 65.15% compared to Edge List’s 52.55%. Although the above results are not better than GNNs according to the reports in [37], these results demonstrate the advantages of graphlet tokenization.

**Why graphlet tokens works.** We analyze per-graph graphlet token frequencies on both datasets (Figure 6). Two key findings emerge: (1) *Intra-domain differences: different graph classes exhibit distinct graphlet frequency signatures.* On IMDB-BINARY, Class 1 graphs average 5.21 occurrences of  $\langle G5\_21 \rangle$  (5-clique) per graph versus 4.06 in Class 0 graphs. On REDDIT-BINARY, the divergence



**Figure 6: Graph Token Frequency (top 50%) of different graph type in IMDB-Binary and Reddit-Binary.**

is more dramatic: Class 1 graphs average 91.8 occurrences of  $\langle G5\_1 \rangle$  (5-node star) per graph—more than double the 25.4 in Class 0 graphs. (2) *Inter-domain differences: different graph domains exhibit fundamentally different dominant motifs.* IMDB graphs are dominated by dense graphlets ( $\langle G5\_21 \rangle$ ), reflecting densely connected actor collaboration networks. REDDIT graphs are dominated by sparse, tree-like graphlets ( $\langle G5\_1 \rangle$ ,  $\langle G2\_edge \rangle$ ), reflecting the hierarchical reply structures of online discussions. Both findings confirm that graphlet tokens provide directly observable, discriminative features at the token level. An LLM can exploit semantic differences in its input token sequence without needing to reconstruct higher-order patterns from raw edges.

## 8 Conclusion

In this paper, we introduce GraphVulcan, a novel framework that bridges the gap between graph-structured data and Large Language Models through reversible, discrete and semantic-rich graph tokenization. Our core innovation lies in treating graphlets as the atomic units of graph encoding, thereby achieve next-graph token prediction. This graphlet-based vocabulary enables exponential information compression while maintaining complete structural fidelity through lossless, reversible tokenization. **Limitations:** We also acknowledge certain limitations in our work, especially insufficient exploration of real-world graph data of diverse domains. Nevertheless, our work represents a meaningful attempt, establishes that structure-aware discrete tokenization is a foundational paradigm shift toward general-purpose graph-language model. Future work includes designing a graph foundation model based on a graph vocabulary that encompass local structural patterns from diverse domains, enabling a single pre-trained Graph Foundation Model to handle a wide range of graph tasks.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 62476245). This work was also supported by Alibaba Group through Alibaba Innovative Research Program.

## References

- [1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [2] Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024. GraphWiz: An Instruction-Following Language Model for Graph Computational Problems (KDD '24). Association for Computing Machinery, New York, NY, USA, 353–364. doi:10.1145/3637528.3672010
- [3] Runjin Chen, Tong Zhao, Ajay Kumar Jaiswal, Neil Shah, and Zhangyang Wang. 2024. LLaGA: Large Language and Graph Assistant. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 7809–7823.
- [4] Sirui Chen, Jiawei Chen, Sheng Zhou, Bohao Wang, Shen Han, Chanfei Su, Yuqing Yuan, and Can Wang. 2024. SIGformer: Sign-aware graph transformer for recommendation. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*. 1274–1284.
- [5] Luigi Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26 (11 2004), 1367 – 1372. doi:10.1109/TPAMI.2004.75
- [6] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv e-prints* (2024), arXiv–2407.
- [7] Paul Erdős, Alfréd Rényi, et al. 1960. On the evolution of random graphs. *Publications of the* (1960).
- [8] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. [n. d.]. Talk like a Graph: Encoding Graphs for Large Language Models. In *The Twelfth International Conference on Learning Representations*.
- [9] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirog Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
- [10] Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066* (2023).
- [11] Xiaojun Guo, Ang Li, Yifei Wang, Stefanie Jegelka, and Yisen Wang. [n. d.]. G1: Teaching LLMs to Reason on Graphs with Reinforcement Learning. In *The Exploration in AI Today Workshop at ICML 2025*.
- [12] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).
- [13] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).
- [14] TN Kipf. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
- [15] Lecheng Kong, Jiarui Feng, Hao Liu, Chengsong Huang, Jiaxin Huang, Yixin Chen, and Muhan Zhang. 2025. GOfA: A Generative One-For-All Model for Joint Graph Language Modeling. In *The Thirteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=mjblC9hfm>
- [16] Luce le Gorrec and Philip A Knight. 2020. A simple embedding for classifying networks with a few graphlets. In *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 635–642.
- [17] Xin Li, Qizhi Chu, Yubin Chen, Yang Liu, Yaoqi Liu, Zekai Yu, Weize Chen, Chen Qian, Chuan Shi, and Cheng Yang. 2024. GraphTeam: Facilitating large language model-based graph analysis via multi-agent collaboration. *arXiv preprint arXiv:2410.18032* (2024).
- [18] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *Advances in neural information processing systems* 36 (2023), 34892–34916.
- [19] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.
- [20] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. TUDataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663* (2020).
- [21] Hang Ni, Jindong Han, Nengjun Zhu, and Hao Liu. 2025. Unsupervised Graph Anomaly Detection via Multi-Hypersphere Heterophilic Graph Learning. *arXiv preprint arXiv:2503.12037* (2025).
- [22] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.
- [23] Nataša Pržulj, Derek G Corneil, and Igor Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508–3515.
- [24] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems* 36 (2023), 53728–53741.
- [25] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (Virtual Event, CA, USA) (KDD '20)*. Association for Computing Machinery, New York, NY, USA, 3505–3506. doi:10.1145/3394486.3406703
- [26] Paul K Rubenstein, Chulayuth Asawaroenchai, Duc Dung Nguyen, Ankur Bapna, Zalan Borsos, Félix de Chaumont Quirry, Peter Chen, Dalia El Badawy, Wei Han, Eugene Kharitonov, et al. 2023. Audiopalm: A large language model that can speak and listen. *arXiv preprint arXiv:2306.12925* (2023).
- [27] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300* (2024).
- [28] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*. PMLR, 488–495.
- [29] Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Long Xia, Dawei Yin, and Chao Huang. 2024. Higt: Heterogeneous graph language model. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*. 2842–2853.
- [30] Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. [n. d.]. GraphArena: Evaluating and Exploring Large Language Models on Graph Computation. In *The Thirteenth International Conference on Learning Representations*.
- [31] Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023. Can Language Models Solve Graph Problems in Natural Language?. In *Advances in Neural Information Processing Systems*, A. Oh, T. N. Naumann, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 30840–30861.
- [32] Jianing Wang, Junda Wu, Yupeng Hou, Yao Liu, Ming Gao, and Julian McAuley. 2024. InstructGraph: Boosting Large Language Models via Graph-centric Instruction Tuning and Preference Alignment. In *Findings of the Association for Computational Linguistics ACL 2024*. 13492–13510.
- [33] Zehong Wang, Zheyuan Zhang, Nitesh Chawla, Chuxu Zhang, and Yanfang Ye. 2024. Gft: Graph foundation model with transferable tree vocabulary. *Advances in Neural Information Processing Systems* 37 (2024), 107403–107443.
- [34] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of ‘small-world’ networks. *nature* 393, 6684 (1998), 440–442.
- [35] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).
- [36] Boris Weisfeiler and Andrei Leman. 1968. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series 2*, 9 (1968), 12–16.
- [37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=ryG6iA5K>
- [38] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).
- [39] Haonan Yuan, Qingyun Sun, Junhua Shi, Xingcheng Fu, Bryan Hooi, Jianxin Li, and Philip S Yu. 2025. GRAVER: Generative Graph Vocabularies for Robust Graph Foundation Models Fine-tuning. *arXiv preprint arXiv:2511.05592* (2025).
- [40] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in neural information processing systems* 32 (2019).
- [41] Mengmei Zhang, Mingwei Sun, Peng Wang, Shen Fan, Yanhu Mo, Xiaoxiao Xu, Hong Liu, Cheng Yang, and Chuan Shi. 2024. Graphtranslator: Aligning graph model to large language model for open-ended tasks. In *Proceedings of the ACM Web Conference 2024*. 1003–1014.
- [42] Qifan Zhang, Xiaobin Hong, Jianheng Tang, Nuo Chen, Yuhan Li, Wenzhong Li, Jing Tang, and Jia Li. 2024. Gocder: Improving large language model for generalized graph problem solving. *arXiv preprint arXiv:2410.19084* (2024).
- [43] Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xiaochuang Han, Tianxing He, and Yulia Tsvetkov. 2024. Can LLM Graph Reasoning Generalize beyond Pattern Memorization? *arXiv preprint arXiv:2406.15992* (2024).
- [44] Huachi Zhou, Jiahe Du, Chuang Zhou, Chang Yang, Yilin Xiao, Yuxuan Xie, and Xiao Huang. 2025. Each Graph is a New Language: Graph Learning with LLMs. *arXiv preprint arXiv:2501.11478* (2025).
- [45] Xi Zhu, Haochen Xue, Ziwei Zhao, Wujiang Xu, Jingyuan Huang, Minghao Guo, Qifan Wang, Kaixiong Zhou, Imran Razzak, and Yongfeng Zhang. 2025. Llm as gnn: Graph vocabulary learning for text-attributed graph foundation models. *arXiv preprint arXiv:2503.03313* (2025).
- [46] Yun Zhu, Haizhou Shi, Xiaotang Wang, Yongchao Liu, Yaoke Wang, Boci Peng, Chuntao Hong, and Siliang Tang. 2025. Graphclip: Enhancing transferability in graph foundation models for text-attributed graphs. In *Proceedings of the ACM on Web Conference 2025*. 2183–2197.

## A On the Purpose of Designing GraphVulcan

A natural question is why an LLM should solve classical graph reasoning tasks by directly reading graph structures rather than simply invoking external graph algorithms. Indeed, for many of the tasks considered in this paper, mature and efficient algorithmic solvers already exist, and tool calling is often the simplest practical solution. However, replacing classical graph algorithms is not the goal of GraphVulcan. Instead, we use these tasks as controlled testbeds with clear supervision signals to investigate a more fundamental question: whether graphs can be represented as discrete, reversible, and semantically meaningful graphlet-token sequences that are compatible with the next-token prediction paradigm of LLMs, in a manner analogous to natural language. From this perspective, GraphVulcan should be viewed as an early proof-of-concept rather than an out-of-the-box downstream system. Its value lies in validating the feasibility of a graph language based on graphlets, showing that LLMs can compose, interpret, and reason over graph structures in token space. We hope future work will extend this paradigm to concrete domains and develop general-purpose graph foundation models capable of structural reasoning.

## B Graph Vocabulary

We present full graph vocabulary with all 30 non-isomorphic graphlets with node count  $2 \leq k \leq 5$ .

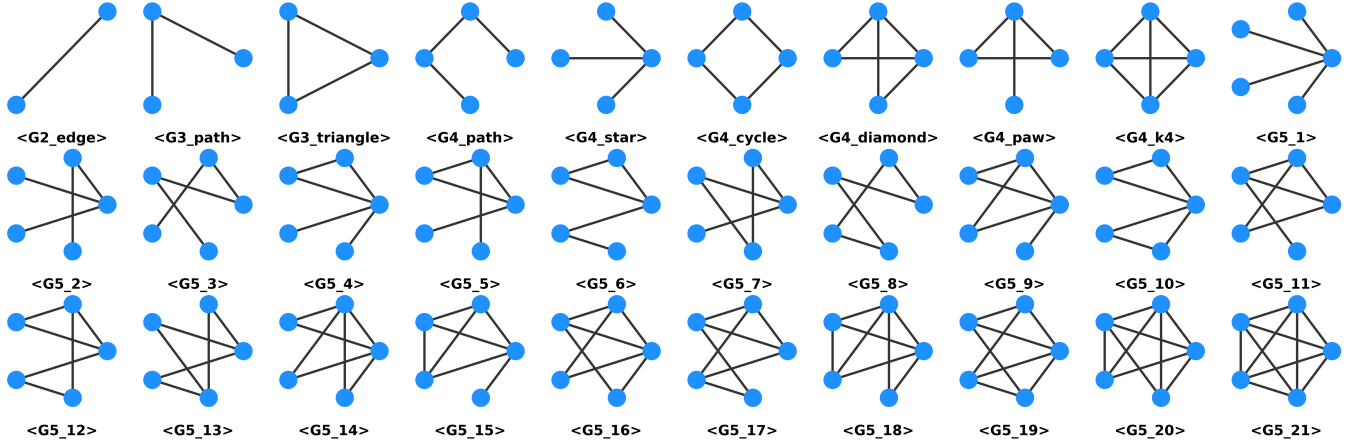


Figure 7: Graph vocabulary containing all 30 non-isomorphic graphlets with node count  $2 \leq k \leq 5$ .

## C Algorithm

### Algorithm 1: Structure-Aware Graph Tokenization

**Require:** Graph  $G = (V, E)$ , maximum graphlet size  $k_{\max}$ , Graph Vocabulary  $\mathcal{V}$   
**Ensure:** Token sequence  $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$

- 1:  $\mathcal{T} \leftarrow \emptyset$  ▷ Initialize empty token sequence
- 2:  $C \leftarrow \text{ConnectedComponents}(G)$
- 3: **for** each component  $G_c \in C$  **do**
- 4:  $G_{\text{rem}} \leftarrow G_c$  ▷ Initialize remaining graph
- 5: **Step 1: Decompose into Graphlets**
- 6: **while**  $E(G_{\text{rem}}) \neq \emptyset$  **do**
- 7:  $V_{\text{sub}} \leftarrow \text{GreedyExpansion}(G_{\text{rem}}, k_{\max})$  ▷ Algorithm 2
- 8:  $G_{\text{sub}} \leftarrow G_{\text{rem}}[V_{\text{sub}}]$  ▷ Induced subgraph
- 9: **Step 2: Match to Vocabulary**
- 10:  $h_{\text{WL}} \leftarrow \text{WeisfeilerLehmanHash}(G_{\text{sub}})$
- 11:  $g_{\text{token}} \leftarrow \text{VocabularyLookup}(h_{\text{WL}})$
- 12:  $\mathcal{T} \leftarrow \mathcal{T} \cup \{g_{\text{token}}\}$  ▷ Add graphlet token
- 13: **Step 3: Update Remaining Graph**
- 14:  $G_{\text{rem}} \leftarrow G_{\text{rem}} \setminus E(G_{\text{sub}})$  ▷ Remove matched edges
- 15: **if**  $E(G_{\text{rem}}) \neq \emptyset$  **then**
- 16:  $\mathcal{T} \leftarrow \mathcal{T} \cup \{\langle G_{\text{Connect}} \rangle\}$  ▷ Add separator token
- 17: **end if**
- 18: **end while**
- 19:  $\mathcal{T} \leftarrow \mathcal{T} \cup \{\langle G_{\text{Disconnect}} \rangle\}$  ▷ Component separator
- 20: **end for**
- 21: **return**  $\mathcal{T}$

### Algorithm 2: GreedyExpansion

**Require:** Remaining graph  $G_{\text{rem}} = (V, E)$ , maximum size  $k_{\max}$   
**Ensure:** Subgraph node set  $V_{\text{sub}} \subseteq V$

- 1: **Step 1: Compute Node Signatures**
- 2:  $\sigma \leftarrow \text{WeisfeilerLehmanColors}(G_{\text{rem}})$  ▷ WL node colors
- 3:  $c \leftarrow \text{ClusteringCoefficient}(G_{\text{rem}})$  ▷ Local clustering
- 4: **Step 2: Select Seed Node**
- 5:  $v_{\text{seed}} \leftarrow \arg \max_{v \in V} (c(v), \sigma(v))$
- 6:  $V_{\text{sub}} \leftarrow \{v_{\text{seed}}\}$
- 7: **Step 3: Greedy Expansion**
- 8: **while**  $|V_{\text{sub}}| < k_{\max}$  **do**
- 9:  $V_{\text{cand}} \leftarrow \mathcal{N}_{G_{\text{rem}}}(V_{\text{sub}}) \setminus V_{\text{sub}}$  ▷ Neighbor candidates
- 10: **if**  $V_{\text{cand}} = \emptyset$  **then**
- 11: **break** ▷ No more neighbors to add
- 12: **end if**
- 13: **Step 4: Score Candidates**
- 14: **for** each  $u \in V_{\text{cand}}$  **do**
- 15:  $s_{\text{density}}(u) \leftarrow |\{(u, v) \in E : v \in V_{\text{sub}}\}|$  ▷ Density score
- 16: **end for**
- 17: **Step 5: Select Best Candidate**
- 18:  $u^* \leftarrow \arg \max_{u \in V_{\text{cand}}} (s_{\text{edge}}(u), c(u), \sigma(u))$
- 19:  $V_{\text{sub}} \leftarrow V_{\text{sub}} \cup \{u^*\}$
- 20: **end while**
- 21: **return**  $V_{\text{sub}}$

## D Training Data Examples

Tables 4 present representative examples of user messages and assistant responses for all seven graph reasoning tasks evaluated in our experiments. These examples demonstrate the input-output format used during training and evaluation, where graphs are encoded using our GraphVocab representation and the model generates step-by-step reasoning chains followed by final answers.

Table 4: Training Data Examples from Graph Reasoning Tasks

Task	User Message	Assistant Response
Connectivity	Given the following graph: <NidB>696<NidS>1<NidS>71<NidS>3<NidS>68<NidE><G5_10><G_Connect><NidB>92<NidS>87<NidS>88<NidS>945<NidS>71<NidE><G5_17><G_Connect><NidB>3<NidS>92<NidE><G2_edge><G_Connect><NidB>46<NidS>87<NidE><G2_edge><G_Connect><NidB>1<NidS>88<NidS>464<NidS>945<NidE><G4_path><G_Disconnect>. Is node 464 and node 1 in a same connected component? Use yes or no to answer.	<think> Node 464 is in the graph token: <NidB>1<NidS>88<NidS>464<NidS>945<NidE><G4_path>. The graph token sequence only has one '<G_Disconnect>' at the end, so every token belongs to the same connected component. Therefore all nodes, including 464 and 1, share the same component. So the answer is Yes. </think> The answer is Yes.
Cycle Detection	Given the following graph: <NidB>24<NidE><G1><G_Disconnect><NidB>204<NidS>4<NidS>43<NidS>88<NidS>0<NidE><G5_6><G_Connect><NidB>88<NidS>573<NidS>3<NidS>0<NidS>73<NidE><G5_3><G_Connect><NidB>43<NidS>69<NidE><G2_edge><G_Disconnect><NidB>5<NidE><G1><G_Disconnect>. Does the graph contain a cycle? Use Yes or No as the final answer.	<think> <NidB>204<NidS>4<NidS>43<NidS>88<NidS>0<NidE><G5_6><G_Operator_Eq><NidB>4<NidS>43<NidS>204<NidE><G3_triangle><G_Connect><NidB>204<NidS>88<NidS>0<NidE><G3_path>. <NidB>4<NidS>43<NidS>204<NidE><G3_triangle>is a cycle visiting nodes 4 ->43 ->204 ->4. So this graph has a cycle. </think> The answer is Yes.
Degree	Given the following graph: <NidB>83<NidS>2<NidS>1<NidS>79<NidS>19<NidE><G5_12><G_Connect><NidB>79<NidS>213<NidS>97<NidS>19<NidE><G5_12><G_Connect><NidB>942<NidS>6<NidS>18<NidS>2<NidS>83<NidE><G5_3><G_Connect><NidB>6<NidS>213<NidE><G2_edge><G_Disconnect><NidB>337<NidE><G1><G_Disconnect>. What is the degree of node 79 in this graph? Your answer should be in the format: 'The degree of node 79 is X.'	<think> I need to calculate the degree of node 79 in the entire graph. First, I will search through the graph token sequence for all graph tokens that contain node 79. These graph tokens are: <NidB>83<NidS>2<NidS>1<NidS>79<NidS>19<NidE><G5_12><G_Connect><NidB>79<NidS>213<NidS>97<NidS>19<NidE><G5_12>. In subgraph <NidB>83<NidS>2<NidS>1<NidS>79<NidS>19<NidE><G5_12>, the degree of node 79 is 2. In subgraph <NidB>79<NidS>213<NidS>97<NidS>942<NidS>19<NidE><G5_12>, the degree of node 79 is 3. obtaining several local degrees. Then, by summing up all these local degrees, I obtain the total degree of node 79 in the entire graph. After calculation, the sum is 5. The degree of node 79 is 5. </think> The degree of node 79 is 5.
Shortest Path	Given the following graph: <NidB>0<NidS>764<NidS>972<NidS>940<NidS>437<NidE><G5_17><G_Connect><NidB>5<NidS>375<NidS>95<NidS>940<NidS>1<NidE><G5_17><G_Connect><NidB>67<NidS>5<NidE><G2_edge><G_Connect><NidB>841<NidS>1<NidS>95<NidS>0<NidS>437<NidE><G5_2><G_Disconnect>. What is the shortest path length between node 95 and node 764? Your answer should be in the format: 'The shortest path length between node 95 and node 764 is X.'	<think> Both nodes are in the same component because '<G_Disconnect>' does not separate them. I will compute the shortest path between node 95 and node 764 by checking all possible shortest paths. <NidB>5<NidS>375<NidS>95<NidS>940<NidS>1<NidE><G5_17><G_Operator_Eq><NidB>940<NidS>5<NidS>95<NidE><G3_path>... and <NidB>940<NidS>5<NidS>95<NidE><G3_path>is a path between nodes 95 and 940, the distance is 2. <NidB>0<NidS>764<NidS>972<NidS>940<NidS>437<NidE><G5_17><G_Operator_Eq><NidB>940<NidS>764<NidE><G2_edge>... and <NidB>940<NidS>764<NidE><G2_edge>is a path between nodes 940 and 764, the distance is 1. All these segments form the path 1: 95 ->5 ->940 ->764. The total distance for this path is 3. But wait, I found another possible path, let's check that next. .... I think there's no more paths to check. After checking all 3 path(s), the shortest path length is: 3. Therefore, the shortest path length between node 95 and node 764 is 3. </think> The shortest path length between node 95 and node 764 is 3.
Isomorphism	Given the following two graphs: G1: <NidB>4<NidS>416<NidS>739<NidS>83<NidS>41<NidE><G5_17><G_Connect><NidB>460<NidS>41<NidS>739<NidS>83<NidE><G4_star><G_Disconnect> and G2: <NidB>8<NidS>15<NidS>7<NidS>262<NidS>4<NidE><G5_17><G_Connect><NidB>6<NidS>7<NidS>4<NidS>262<NidE><G4_star><G_Disconnect>. Are the two graphs isomorphic? Use Yes or No to answer.	<think> To determine if the two graphs are isomorphic, we can follow these steps: <NidB>4<NidS>416<NidS>739<NidS>83<NidS>41<NidE><G5_17> matches with <NidB>8<NidS>15<NidS>262<NidS>7<NidS>4<NidE><G5_17>. <NidB>460<NidS>739<NidS>83<NidS>41<NidE><G4_star> matches with <NidB>6<NidS>7<NidS>4<NidS>262<NidE><G4_star>. All components successfully matched, therefore they are isomorphic. </think> The answer is Yes.
Max Clique	You are required to solve the Maximum Clique problem. Your goal is to identify the largest complete subgraph (clique) in the given graph. You are given the following graph: <NidB>876<NidS>131<NidS>676<NidS>918<NidS>7<NidE><G5_1><G_Disconnect>. Provide the indices of the nodes in the maximum clique in the following format: The maximum clique is: [Node indices]. For example, if the maximum clique consists of nodes 1, 3, 5, you should answer: The maximum clique is: [1, 3, 5].	<think> To find the Maximum Clique, we can decompose graph tokens as follows: <NidB>876<NidS>131<NidS>676<NidS>918<NidS>7<NidE><G5_1><G_Operator_Eq><NidB>131<NidS>876<NidE><G2_edge><G_Connect><NidB>876<NidS>676<NidS>918<NidS>7<NidE><G4_star>. No more tokens can be further decomposed to form a maximum clique. We can see that the maximum clique is formed by the following token sequence: <NidB>131<NidS>876<NidE><G2_edge>. Thus, the maximum clique consists of nodes: [131, 876]. </think> The maximum clique is: [131, 876]
Max Common Subgraph	You are required to solve the Maximum Common Subgraph problem. Your goal is to identify the common subgraph with the maximum number of nodes shared between the two graphs. You are given the following two graphs: Graph A: <NidB>471<NidS>3<NidS>393<NidS>2<NidS>419<NidE><G5_5><G_Connect><NidB>96<NidS>393<NidE><G2_edge><G_Connect><NidB>2<NidS>65<NidS>419<NidE><G3_path><G_Disconnect> Graph B: <NidB>296<NidS>316<NidS>5<NidS>517<NidS>4<NidE><G5_6><G_Connect><NidB>141<NidS>316<NidE><G2_edge><G_Disconnect> Provide the indices of the nodes in the common subgraph for each graph in the following format: Graph A: [Node indices in graph A], Graph B: [Node indices in graph B]. For example, if the common subgraph is the subgraph of node 1, 2, 3 in graph A and the subgraph of node 2, 3, 4 in graph B, you should answer: Graph A: [1, 2, 3], Graph B: [2, 3, 4].	<think> To find the Maximum Common Subgraph (MCS) between the two graphs, we can decompose and match graph tokens as follows: <NidB>3<NidS>471<NidS>393<NidS>419<NidS>2<NidE><G5_5><G_Operator_Eq><NidB>393<NidS>471<NidS>3<NidS>2<NidE><G4_paw>... <NidB>2<NidS>65<NidS>419<NidE><G3_path><G_Operator_Eq><NidB>2<NidS>65<NidE><G2_edge>... The decomposed tokens can be recomposed as <NidB>471<NidS>3<NidS>393<NidS>2<NidS>65<NidE><G5_6>. <NidB>471<NidS>3<NidS>393<NidS>2<NidS>65<NidE><G5_6> matches with <NidB>296<NidS>316<NidS>5<NidS>517<NidS>4<NidE><G5_6>. <NidB>96<NidS>393<NidE><G2_edge> matches with <NidB>141<NidS>316<NidE><G2_edge>. The remaining graph <NidB>3<NidS>419<NidE><G2_edge>, <NidB>65<NidS>419<NidE><G2_edge> could not be matched. Thus the maximum common subgraph maps nodes [2, 3, 65, 96, 393, 471] in Graph A to nodes [4, 5, 141, 296, 316, 517] in Graph B. </think> Graph A: [2, 3, 65, 96, 393, 471], Graph B: [4, 5, 141, 296, 316, 517]