OpenGraph: Towards Open Graph Foundation Models

Lianghao Xia
University of Hong Kong
aka_xia@foxmail.com

Ben Kao
University of Hong Kong
kao@cs.hku.hk

Chao Huang[*]
University of Hong Kong
chuang7@hku.hk

Abstract

Graph learning has become essential in various domains, including recommendation systems and social network analysis. Graph Neural Networks (GNNs) have emerged as promising techniques for encoding structural information and improving performance in tasks like link prediction and node classification. However, a key challenge remains: the difficulty of generalizing to unseen graph data with different properties. In this work, we propose a novel graph foundation model, called OpenGraph, to address this challenge. Our approach tackles several technical obstacles. Firstly, we enhance data augmentation using a large language model (LLM) to overcome data scarcity in real-world scenarios. Secondly, we introduce a unified graph tokenizer that enables the model to generalize effectively to diverse graph data, even when encountering unseen properties during training. Thirdly, our developed scalable graph transformer captures node-wise dependencies within the global topological context. Extensive experiments validate the effectiveness of our framework. By adapting OpenGraph to new graph characteristics and comprehending diverse graphs, our approach achieves remarkable zero-shot graph learning performance across various settings. We release the model implementation at https://github.com/HKUDS/OpenGraph.

1 Introduction

Graph learning is a crucial methodology in various fields, such as recommender systems (He et al., 2020), social network analysis (Sankar et al., 2021), citation networks (Lv et al., 2021), and transportation networks (Wang et al., 2020). By utilizing Graph Neural Networks (GNNs) and recursive message passing, we capture the complex structures of graphs effectively. GNNs leverage interdependencies among nodes to incorporate high-order connectivities into learned graph representations (Ying et al., 2018; Jin et al., 2020).

A primary challenge in current end-to-end graph neural networks is their heavy reliance on scarce and low-quality labeled data (Liu et al., 2022; Jin et al., 2022). To overcome this, self-supervised learning (SSL) has emerged as a solution by leveraging augmented self-supervision signals. Contrastive SSL, exemplified by DGI (Veličković et al., 2018b) and GraphCL (You et al., 2020), incorporates contrastive objectives as self-supervised alignment loss. Recent advancements like JOAO (You et al., 2021) and GCA (Zhu et al., 2021) automate contrastive learning through adaptive augmentation. By integrating SSL techniques, we enhance graph neural networks with limited labeled data.

Graph pre-training excels at capturing intrinsic graph properties but struggle to effectively generalize to diverse downstream domains, particularly when faced with distribution shifts (Sun et al., 2023; Wu et al., 2021c; Gui et al., 2022). For example, in recommender systems, handling previously unseen user interaction graphs in cold-start recommendation scenarios is crucial (Chen et al., 2022a). Transferring knowledge from pre-trained graph domains to other downstream domains is desirable (Zhang et al., 2022a). However, applying these models to unseen graphs results in significant performance deterioration due to variations in node sets and relation semantics across different scenarios.

Recent research explores prompt-tuning as a task-specific alternative to fine-tuning, bridging the gap between pre-training and downstream objectives (Sun et al., 2022; Liu et al., 2023; Fang et al., 2023). These approaches align the pre-trained model's understanding with specific task requirements. However, practical scenarios involve variations in node sets and feature semantics across diverse downstream graph domains. Further exploration is needed to enhance graph models' generalization and adaptability to real-world graphs.

[*]Chao Huang is the corresponding author.

This work aims to develop a scalable graph model that enables zero-shot learning, effectively making accurate predictions on unseen graphs. Building such a model poses significant challenges.

- **C1**: **Domain-Specific Data Scarcity**. Data scarcity is a common challenge across downstream domain tasks, driven by factors like privacy concerns. Limited availability of domain-specific user behavior graphs restricts data collection. Therefore, developing label-less learning frameworks within graph models is crucial to effectively understand the context of downstream tasks in the face of data scarcity.
- **C2**: **Node Token Set Shift**. A key challenge in zero-shot graph learning is the shift in node token sets across graphs. This requires the model to reconcile variations in node characteristics. Generating universal graph tokens is crucial to effectively represent and comprehend diverse unseen graphs with different topological contexts.
- **C3**: **Efficient Graph Dependency Modeling**. Nodes in large-scale graphs have complex dependencies. Understanding local and global interdependencies among all nodes is crucial for accurate prediction. Efficient node-wise dependency encoding is vital to enhance the performance and scalability of graph models.

Present Work. To overcome the challenges, we introduce a graph model for zero-shot learning that captures universal and transferable structural patterns across multiple domains. To address **C1**, we propose combining large language models (LLMs) with data augmentation techniques for synthetic graph generation. By generating augmented graphs resembling real-world instances, we enhance the pre-training process of OpenGraph and gain a deeper understanding of downstream task contexts. This is achieved through the integration of tree-of-prompt regularization with Gibbs sampling.

To address **C2**, we propose a topology-aware graph tokenizer that generates universal graph tokens from arbitrary graphs. For **C3**, we develop a scalable graph transformer with efficient self-attention using anchor sampling. Our approach ensures computational efficiency through a two-stage self-attention process and optimizes training by leveraging token sequence sampling, reducing sequence length while preserving crucial context. We conducted extensive experiments on diverse datasets, demonstrating the remarkable generalization abilities of our model across various settings.

2 Preliminaries

Graph Learning. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{F})$ consists of a node set $\mathcal{V} = v_i$, an edge set $\mathcal{E} = (v_s, v_t)$, and node attributes $\mathbf{F} \in \mathbb{R}^{|\mathcal{V}|\times f}$. Graph learning aims to produce node representations that encode both structural and attribute information. These embeddings are used for tasks such as link prediction and node classification, involving the prediction of node connections and categories, respectively. The corresponding losses to be minimized are:

$$\mathcal{L}_{\text{link}} = \sum_{v_s, v_t} (1 - e_{s,t}) f(v_s, v_t) - e_{s,t} f(v_s, v_t),$$
$$\mathcal{L}_{\text{node}} = -\sum_{v_s, y_s} \Big(f(v_s, y_s) \;/ \sum_{y_s' \neq y_s} f(v_s, y_s') \Big) \quad (1)$$

where $e_{s,t} \in \{0, 1\}$ denotes the link label for nodes v_s and v_t, and $y_s \in \mathcal{C}$ indicate the groundtruth category for node v_s. Function f denotes the prediction model with learnable parameters $\mathbf{\Theta}_f$.

Zero-shot Graph Learning. Current graph models excel in standard tasks but struggle to generalize across diverse domains. Their performance deteriorates when applied to new graphs with varying characteristics, such as node sets and features. To address these limitations, we focus on *zero-shot graph learning*, where a model is trained on a set of graphs and evaluated on different test graphs without shared graph tokens. It aims to assess the model's ability to learn generalized topological structures and node-wise dependencies. Formally, we seek to minimize the error measurement $\epsilon(\mathcal{G}_t, f)$, with $\arg\min_f$ denoting the optimization.

$$\mathbf{\Theta}_f = \arg\min\nolimits_{\mathbf{\Theta}_f} \mathcal{L}(\{\mathcal{G}_s\}, f),$$
$$\mathcal{V}_t \cap \mathcal{V}_s = \varnothing,\ \mathcal{E}_t \cap \mathcal{E}_s = \varnothing,\ \mathbb{R}^{f_t} \neq \mathbb{R}^{f_s} \quad (2)$$

This objective is to develop a universal graph modeling architecture $f(\cdot)$. Its parameters $\mathbf{\Theta}_f$ are learned by minimizng graph learning losses $\mathcal{L}$ on the training graphs $\{\mathcal{G}_s\}$. Notably, the training graphs and test graphs $\mathcal{G}_t$ have no common nodes, edges, or node features. This presents a unique challenge for the graph model to handle the significant distribution shift that occurs across different graph domains with entirely distinct datasets.

3 Methodology

This section presents the design details of the proposed OpenGraph framework. Figure 1 gives an overall illustration for the model. In appendix, we elaborate on how OpenGraph handles node classification (A.1.1) and graph features (A.1.2), detailed configurations of our scalable graph transformer (A.1.3), as well as the generation algorithms (A.2).

3.1 Unified Graph Tokenizer

To handle diverse graphs with varying nodes and features, our goal is to develop a graph tokenizer that transforms input graphs into unified token sequences: $\mathcal{G} \rightarrow \{\mathbf{e}_i\}$. Each token represents a node accompanied by a semantic vector $\mathbf{e}_i$. By utilizing a shared representation space and a flexible sequence structure, we aim to standardize distributions across graphs. Specifically, our tokenizer uses the smoothed adjacency matrix $\tilde{\mathbf{A}}$, and a topology-aware projection function $\phi : \mathbb{R}^{|\mathcal{V}|} \rightarrow \mathbb{R}^d$.

3.1.1 Smoothed High-Order Adjacency

We start with the original adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ built from edges $\mathcal{E}$. The smoothing procedure for the adjacency matrix is as follows:

$$\tilde{\mathbf{A}} = \bar{\mathbf{A}}^1 + \bar{\mathbf{A}}^2 + \cdots \bar{\mathbf{A}}^L, \quad \bar{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \quad (3)$$

For numerical stability, we use Laplacian normalization $\bar{\mathbf{A}}$ with the diagonal degree matrix $\mathbf{D}$ of adjacency $\mathbf{A}$. To capture high-order connectivity and sparse node-wise relations, OpenGraph combines $\bar{\mathbf{A}}$ at different orders. This provides us with topology information for further processing, with L representing the maximum power order considered.

3.1.2 Topology-aware Projection with Arbitary Graphs

To handle the varying dimensionality $|\mathcal{V}| \times |\mathcal{V}|$ of adjacency $\tilde{\mathbf{A}}$, OpenGraph applies a projection function $\phi : \mathbb{R}^{|\mathcal{V}|} \rightarrow \mathbb{R}^d$ to transform the adjacency into sequence data. A large hidden dimensionality d is used to minimize information loss. Previous research has shown that even random projections with large dimensions can achieve satisfactory performance (Zheng et al., 2022). To preserve topology information, we employ fast singular value decomposition (SVD) as the projection ϕ. SVD is known for its efficiency and effectiveness in adjacency compression (Jamali and Ester, 2010). Our empirical analysis demonstrates that two iterations of fast SVD effectively preserve topology information with minimal computational overhead. The graph tokenizer performs the following operations to calculate the resulting token sequence:

$$\mathbf{e}_v = \phi(\tilde{\mathbf{A}}_{v,:}) = \tilde{\mathbf{A}}_{v,:} \cdot \mathrm{LN}((\mathbf{U}\sqrt{\Lambda} \parallel \mathbf{V}\sqrt{\Lambda})) \quad (4)$$

where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{|\mathcal{V}| \times d}$ and $\Lambda \in \mathbb{R}^{d \times d}$ are obtained from SVD. The concatenation operator $\parallel$ combines them in the hidden dimension. Layer normalization function $\mathrm{LN}(\cdot)$ reduces numerical variance across datasets. The resulting $\mathbf{e}_v \in \mathbb{R}^d$ incorporates topology information from $\tilde{\mathbf{A}}$ and the topology-aware projection ϕ. This information strengthens subsequent learnable neural networks.

3.2 Scalable Graph Transformer

With the universal topology-aware graph tokens, the subsequent task is to empower our graph model to grasp the complex node-wise dependencies within the global context. Inspired by the success of transformer architectures in modeling complex relationships between instances, OpenGraph utilizes a graph transformer as the backbone. To ensure scalability and effectiveness for large-scale graphs, we introduce the following techniques.

Token Sequence Sampling. For efficiency, we train the graph transformer using sampled token sequences from the current training batch, which contains centric nodes v_{c_b}, positive nodes v_{p_b}, and negative nodes v_{n_b}. The input is as follows:

$$(\mathbf{e}_{c_1} \cdots \mathbf{e}_{c_B}) \parallel (\mathbf{e}_{p_1} \cdots \mathbf{e}_{p_B}) \parallel (\mathbf{e}_{n_1} \cdots \mathbf{e}_{n_B}) \quad (5)$$

This approach significantly reduces the sequence length from $|\mathcal{V}|$ to $3 \times B$, enabling efficient training for large-scale graphs. Despite using a subsequence, the topology-aware embeddings contain local structural information for each node and reflect the overall graph structure. Additionally, this sampling technique emphasizes the current training batch, leading to further training improvements.

Efficient Self-Attention with Anchors. To accelerate the self-attention part of OpenGraph with quadratic complexity, we introduce a step of sampling anchor nodes v_{a_s} for $s \in S$, where $S < 3B$. This splits the self-attention process into two stages: propagating messages from all nodes to the anchor nodes and then propagating the anchor embeddings to all nodes. This decomposition reduces the complexity from $\mathcal{O}(B^2 \times d)$ to $\mathcal{O}(B \times S)$, ensuring scalability for large-scale graphs.

3.3 Knowledge Distillation from LLM

Obtaining diverse graph datasets for different domains can be challenging due to factors like privacy issues that restrict access to essential data (Zheleva and Getoor, 2007). Inspired by the remarkable knowledge and understanding demonstrated

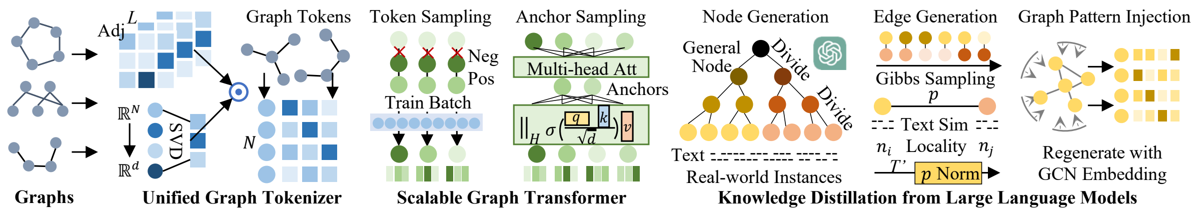


Figure 1: Overall model architecture of the OpenGraph framework.

by large language models (LLMs), we leverage their power to enhance the generation of diverse graph-structured data. To improve the efficacy of our LLM-augmented graph data for pre-training our model, we have developed an augmentation mechanism. This mechanism enables the LLM-augmented graph data to closely approximate real-world graph characteristics, enhancing the relevance and usefulness of the augmented data.

3.3.1 LLM-based Node Generation

Our first step is to create a node set tailored to the application scenario, characterized by text-based profiles that generate subsequent edges. However, dealing with real-world scenarios poses challenges due to the large scale of the node set. For example, e-commerce platforms may have billions of products, making it challenging for the LLM to efficiently generate a large number of nodes.

To address this challenge, we adopt an iterative strategy of dividing general nodes into sub-categories with finer semantic granularity. For instance, in the case of generating product nodes, we prompt the LLM with a query like "List sub-categories of *products* on platforms like Amazon." The LLM provides a list of sub-categories such as "clothing" and "electronics." We repeat this iterative division process, refining each sub-category further, until we obtain nodes that resemble real-world instances, such as "women's clothing," "sweaters," "hooded sweaters," and "white hooded sweaters." Appendix A.2.1 presents details on our prompt template and generation examples.

Tree-of-Prompt Algorithm. The process of dividing nodes into sub-categories and generating fine-grained entities follows a tree structure. The initial general node (e.g., "products," "deep learning papers") serves as the root, and fine-grained entities act as leaf nodes. We employ a tree-of-prompt strategy to traverse and generate these nodes. For further details, please see Appendix A.2.2.

3.3.2 Edge Sampling using Node Profiles

To generate edges, we use the Gibbs sampling algorithm (Gelfand, 2000) with the generated node set $\mathcal{V}$. The algorithm starts with a random sample.For instance, in a paper-wise citation network, the initial sample is a node pair (v_{s_0}, v_{t_0}). In a person-entity relation scenario like an author-paper network, the initial sample is a binary vector $\mathbf{a}^0 \in \{0,1\}^{|\mathcal{V}|}$. Each element $a_i \in \mathbf{a}^0$ indicates whether there is an interaction between the sampled person and the i-th node v_i. In the case of person-entity relations, the Gibbs algorithm for edge sampling is described in Appendix A.2.3. The key is estimating the probability $p(\mathbf{a}^t \oplus v_{t'}|\mathbf{a}^t)$, with $\oplus$ representing setting the t'-th dimension of $\mathbf{a}^t$ to 1.

Node-wise Connection Probability Estimation. To estimate the probability $p(\mathbf{a}^t \oplus v_{t'}|\mathbf{a}^t)$ of connecting two nodes in our generated graph, we leverage the reasoning capabilities of the LLM. However, directly prompting the LLM for predictions on each edge can be computationally expensive, with $\mathcal{O}(|\mathcal{V}| \times |\mathcal{V}|)$ prompts required. To ensure efficiency, we adopt an alternative approach. We prompt the LLM to generate hidden representations $\mathbf{h}_i$ for each node v_i. Then, we calculate the probability for each edge with dot-product as:

$$p(\mathbf{a}^t \oplus v_{t'}|\mathbf{a}^t) = \sum\nolimits_{v_i} a_i^t (\mathbf{h}_i / \|\mathbf{a}^t\|_0)^\top \cdot \mathbf{h}_{t'} \quad (6)$$

By utilizing the text embeddings $\mathbf{h}_i$ and $\mathbf{h}_{t'}$ provided by the LLM, we can effectively capture the semantic relations between the respective nodes.

Dynamic Probability Normalization. To ensure that the calculated probability scores fall within a reasonable range like $[0,1]$, our generation algorithm incorporates a dynamic probability normalization approach.It maintains a record of the most recent T' estimation values, denoted as $\mathcal{P} = \{p(\mathbf{a}^t \oplus v_{t'}|\mathbf{a}^t) \mid t = -1, \cdots, -T'\}$. By calculating the mean (μ) and standard deviation (σ) of these values, we gain insight into their distribution. New estimations are adjusted using $\mu \pm 2\sigma$ as the bounds, resulting in $\bar{p} = (p - \mu)/(4\sigma)$.

Node Locality Incorporation. To address the limitation of the previous edge sampling algorithm based on semantic similarity, we introduce the concept of locality. Each node is assigned a random locality index, and we consider the difference in locality using an exponential decay function. This results in an adjusted probability calculated as $\hat{p} = \bar{p} \cdot \alpha^{|n_i - n_j|}$, where $0 < \alpha < 1$. By incorporating locality, we account for the observed patterns in real-world graphs and prevent excessive connections among semantically-related nodes.

3.3.3 Graph Topological Pattern Injection

To enhance the incorporation of topological information in the graph generation process, we refine the node embeddings after the initial graph generation. By training a Graph Convolutional Network (GCN) on the graph $\mathcal{G}$, we obtain new node embeddings that capture the underlying topology patterns. This aligns the node embeddings derived from the graph with the textual embeddings of the entities and avoids distribution shifts between graph and textual spaces. The final graph is constructed using our edge generation algorithm, which operates on these enhanced node representations.

4 Evaluation

4.1 Experimental Settings

Datasets. We evaluate OpenGraph on two graph learning tasks: link prediction and node classification, using totally 8 real-world datasets. Appendix A.3.1 provides detailed descriptions.

Evaluation Protocol. Following previous works (He et al., 2020; Kipf and Welling, 2017), we adopt the original train-test data split for the experimental datasets. We pre-train our OpenGraph on generated datasets and conducts zero-shot prediction for the evaluation datasets made of real graph data. As most baselines struggle with cross-dataset transferring, we evaluate them in two few-shot training settings. Please refer to Appendix A.3.2 for more details about our cross-dataset zero-shot setting, few-shot settings, and evaluation metrics.

Implementation Details. We provide detailed information about the implementation of OpenGraph and the baseline methods, as well as the graph generation process, in Appendix A.3.3.

Baselines. Our empirical evaluation utilizes the following 9 state-of-the-art baseline methods from 4 different research lines. Detailed descriptions for the baselines can be found in Appendix A.3.4.

4.2 Overall Performance Comparison (RQ1)

Comparing the zero-shot performance of OpenGraph with the few-shot performance of baselines in link prediction and node classification (Table 1), we have the following observations:

Predominant Performance of OpenGraph. Our model outperforms baselines on all 8 datasets in different categories without using any overlapping data between pre-training and downstream tasks, showcasing its remarkable ability to generalize. This advantage can be attributed to three key factors: i) the unified graph tokenizer, bridging the gap between pre-training and target datasets; ii) the scalable graph transformer, capturing important structural features and learning relations effectively; and iii) effective pre-training with LLM-generated graph data, equipping our model with versatile forecasting abilities. Overall, these design choices contribute to our model's outstanding generalization capabilities across diverse datasets.

Limitations of Existing Pretraining Methods. Pre-training methods like GraphCL and DGI do not consistently outperform foundational models (e.g., GIN and GCN) trained on few-shot data, suggesting that they may hinder performance when applied to different datasets. This is due to a significant shift in data distribution between pre-training and target datasets, leading to overfitting and impairing adaptability to new graph structures in downstream tasks. Prompt tuning methods like GraphPrompt and GPF sometimes degrade more severely compared to full-model fine-tuning approaches, indicating a higher susceptibility to overfitting to distinctive patterns within the training set.

Tackling Node Classification by Structure Learning. Our OpenGraph shows significant improvement in node classification tasks, highlighting the effectiveness of using our pre-trained link predictor to identify connections between ordinary nodes and special nodes representing classes. This advantage relies on OpenGraph's strong ability to generalize, transferring knowledge across datasets and tasks. The credit for this versatility goes to the universal applicability and flexibility of our graph tokenization and encoding.

4.3 Investigation on Graph Tokenizer (RQ2)

In this section, we study the effectiveness of our graph tokenizater by evaluating the impact of the smoothed adjacency matrix and comparing our projection method with alternative compression methods. The results are summarized in Figure 2.

Impact of adjacency smoothing: We examine the effect of graph smoothing on model performance by testing different levels of smoothing for the input adjacency matrix. The results are depicted in Figure 2(a) and 2(b). Here, the use of 0 adjacency

Table 1: Performance of OpenGraph (zero-shot) and baseline methods (one-shot, five-shot) on link prediction (measured by *Recall@N* for $N = 20, 40$) and node classification (measured by *Accuracy* and *Macro F1 Score*).

Dataset		ogbl-ddi		ogbl-collab		ML-1M		ML-10M		Amazon-book		Cora		Citeseer		Pubmed	
Model	shot	R-20	R-40	R-20	R-40	R-20	R-40	R-20	R-40	R-20	R-40	Acc	F1	Acc	F1	Acc	F1
MF	1	.0087	.0161	.0261	.0349	.0331	.0604	.1396	.1956	.0034	.0043	.1710	.1563	.1740	.1727	.3470	.3346
	5	.0536	.0884	.0412	.0609	.0987	.1584	.2060	.2989	.0196	.0284	.1500	.1422	.1520	.1484	.3540	.3435
MLP	1	.0195	.0336	.0112	.0185	.0548	.1019	.1492	.2048	.0017	.0028	.2300	.1100	.2590	.1993	.4430	.3114
	5	.0621	.1038	.0115	.0185	.0851	.1470	.2362	.2563	.0092	.0152	.3930	.3367	.3690	.3032	.5240	.4767
GCN	1	.0279	.0459	.0206	.0321	.0432	.0849	.1760	.2086	.0096	.0160	.3180	.1643	.3200	.2096	.4270	.3296
	5	.0705	.1312	.0366	.0513	.1054	.1656	.2127	.2324	.0251	.0408	.5470	.5008	.4910	.4190	.509	.4455
GAT	1	.0580	.1061	.0258	.0372	.0245	.0520	.1615	.2476	.0047	.0079	.2420	.1687	.2810	.2025	.4720	.3657
	5	.0711	.1309	.0340	.0505	.1506	.2267	.2002	.2883	.0228	.0392	.585	.5438	.4940	.4441	.5780	.5582
GIN	1	.0530	.1004	.0163	.0247	.0466	.0884	.1541	.2388	.0069	.0114	.3190	.1753	.2820	.1705	.4410	.3064
	5	.0735	.1441	.0311	.0458	.1458	.2344	.1926	.2829	.0252	.0418	.5400	.4941	.521	.4696	.5070	.4547
DGI	1	.0315	.0617	.0255	.0385	.0486	.0863	.1868	.2716	.0081	.0142	.3150	.1782	.2840	.1791	.4290	.3163
	5	.0821	.1426	.0345	.0502	.1687	.2573	.2303	.3063	.0300	.0492	.4880	.4606	.4450	.4062	.4890	.4509
GPF	1	.0503	.0856	.0027	.0048	.1099	.1702	.1599	.2326	.0072	.0128	.3080	.1952	.3110	.1984	.4220	.2670
	5	.0839	.1460	.0027	.0047	.0817	.1392	.2014	.2994	.0179	.0310	.5550	.5233	.4690	.4223	.5150	.4934
GPrompt	1	.0541	.1102	.0138	.0207	.0797	.1310	.1362	.2073	.0074	.0120	.3540	.1596	.2800	.1519	.4710	.3705
	5	.0769	.1396	.0157	.0231	.1340	.2166	.2157	.3147	.0287	.0464	.5510	.5098	.5570	.5211	.5130	.4520
GraphCL	1	.0603	.1112	.0265	.0398	.0390	.0799	.1655	.2529	.0047	.0077	.2430	.1548	.2980	.1630	.4070	.4130
	5	.0740	.1368	.0311	.0456	.1416	.2138	.2019	.3075	.0270	.0440	.5610	.5330	.4300	.3683	.5230	.5024
Ours	0	**.0921**	**.1746**	**.0421**	**.0639**	**.1911**	**.2978**	**.2370**	**.3265**	**.0485**	**.0748**	**.7504**	**.7426**	**.7221**	**.6801**	**.6869**	**.6537**

smoothing implies the input of an identity matrix for the graph tokenizer. This approach significantly damages the topological information for the graph tokenizer, resulting in poor performance. This outcome underscores the importance of considering the adjacency matrix within our unified graph tokenizer. For the non-zero graph smoothing orders, $L = 2$ produces the best performance for the Movielens-1M dataset. $L = 3$ and 1 yield the best performance for the OGBL-ddi data under the top-20 and top-40 settings, respectively. This suggests the benefits of exploring high-order graph smoothing in the graph tokenizer of our OpenGraph.

Superiority of topology-aware projection. To assess the effectiveness of our topology-aware projection based on SVD, we compare it to three alternative projection methods (see Appendix A.3.5 for details). The results are presented in Figure 2(c) and 2(d). We make the following observations:

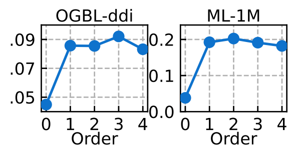
- **One-hot encoding**. This approach learns id-corresponding embeddings across datasets. It performs poorly in the zero-shot evaluation, highlighting the difficulty of transferring dataset-specific parameters like node embeddings to unseen datasets that lack overlapping node tokens.
- **Degree embeddings**. This method learns degree-specific embeddings. It performs significantly worse than our projection scheme. This is because there is a substantial semantic gap for the same degree number across different graphs. Moreover, it oversimplifies topology features by considering only the number of direct links, limiting its ability to capture nuanced structural patterns and adversely affecting graph projection.
- **Random projection**. It randomly assigns unlearnable embedding vectors to nodes. It outperforms the other two variants, but its performance is still inferior to our method due to the low representation efficiency of its uniform distribution.

4.4 Influence of Pre-training Datasets (RQ3)

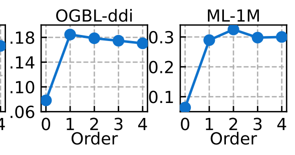
To evaluate the effectiveness of our knowledge distillation from the LLM, we compare the performance of OpenGraph networks pre-trained with different datasets. We use three ablated versions of our graph generation algorithm, namely -Norm, -Loc, and -Topo. Additionally, we incorporate two real datasets, Yelp2018 and Gowalla, for pre-training, which are unrelated to the test datasets. The ML-10M dataset, related to ML-1M and itself, is also included. The evaluation results are summarized in Table 2. We draw the following conclusions:

Superiority of our generated data. Our generated dataset (Gen) achieves the best performance on all test datasets except for ML-1M and ML-10M. Notably, ML-10M, which is closely related to these two datasets, achieves the best performance in these cases. This finding highlights the superior generalization ability of our generated datasets, which equips the OpenGraph model with the capability of universal topological structure learning.

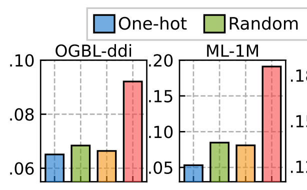
Impact of individual generation techniques. We conduct ablation study by removing the dynamic probability normalization (-Norm), locality incorporation (-Loc), and graph topological pattern in-



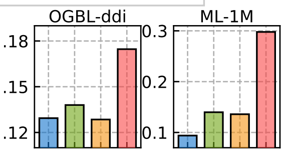
(a) Recall@20



(b) Recall@40



(c) Recall@20



(d) Recall@40

Figure 2: Influence of graph tokenizer configurations.

Table 2: Impact of using different pre-training datasets.

Test Data	Pre-training Dataset						
	-Norm	-Loc	-Topo	Yelp	Gowalla	ML10M	Gen
ogbl-ddi	.0737	.0893	.0656	.0588	.0770	.0692	**.0921**
ML1M	.0572	.1680	.0850	.0599	.0485	**.2030**	.1911
ML10M	.0982	.1636	.1017	.1629	.0910	**.2698**	.2370
Cora	.4985	.4864	.4342	.3715	.5943	.2780	**.7504**
Citeseer	.3944	.3691	.5743	.2651	.4300	.2003	**.7221**
Pubmed	.4501	.5015	.4876	.3317	.5148	.3652	**.6869**

jection (-Topo) modules. The removal of these modules from the generation algorithm leads to a significant drop in performance for the downstream prediction. This demonstrates the importance of these key modules for our generated graphs.

Real-world data can be misleading. Using real-world datasets (Yelp and Gowalla) does not yield transferable graph learning capabilities that provide an advantage over our Gen data. This highlights the limitation of relying solely on insufficient or biased real-world data for cross-data pre-training.

Pre-training on related datasets is useful. Models pre-trained on the ML-10M dataset exhibit superior performance when testing on ML-1M and ML-10M, which are directly related datasets. This indicates that dataset-wise similarity can aid in the cross-dataset knowledge transferring.

4.5 Impact of Sampling in Transformer (RQ4)

We examine the influence of token sequence sampling and anchor sampling in our scalable graph transformer architecture. The metrics for efficiency and performance are summarized in Table 3. Our evaluation focuses on the GPU memory costs and the running time during both the training and testing processes. Additionally, we assess the model performance after end-to-end training. The following observations are made for the ablated versions.

- **-S-A**: It eliminates both token sequence sampling and anchor sampling, leading to out-of-memory

Table 3: Impact of sampling strategies on the efficiency and performance in the scalable graph transformer.

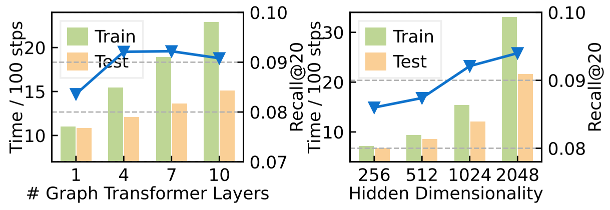
OGBL-ddi	Memory		Time		R@20
	Train	Test	Train	Test	
-S-A	5420MiB	1456MiB	22.72s	13.88s	0.0966
-Anc	3360MiB	1456MiB	18.19s	13.73s	0.1107
-Seq	2456MiB	1202MiB	16.45s	12.09s	0.0930
Ours	2358MiB	1202MiB	15.45s	12.09s	0.1006
ML-10M	Memory		Time		R@20
	Train	Test	Train	Test	
-S-A	OOM	OOM	–	–	–
-Anc	4996MiB	OOM	73.15s	–	–
-Seq	23140MiB	4550MiB	158.60s	84.78s	0.2772
Ours	4470MiB	4550MiB	68.79s	54.17s	0.2816

(OOM) errors when applied to the larger ML-10M data. When evaluated on OGBL-ddi, it exhibits the lowest memory and time efficiency, while its performance is inferior to OpenGraph.
- **-Anc**: This version incorporates only sequence sampling. It significantly reduces memory costs during the training phase. Additionally, by focusing on the current training context, it achieves the best performance on OGBL-ddi.
- **-Seq**: This model removes token sequence sampling from OpenGraph, resulting in a significant decrease in training efficiency. However, the anchor sampling strategy in this version greatly reduces computational costs during the test phase. Despite the computational benefits, the anchor sampling strategy leads to a drop in performance compared to the full-version OpenGraph.

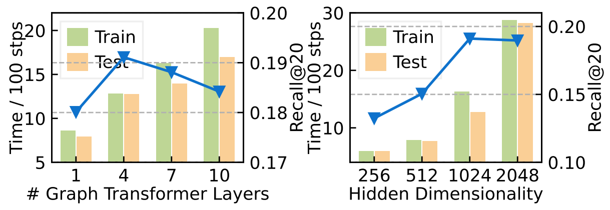
4.6 Impact of Model Scale (RQ5)

This section examines the influence of model scale within our OpenGraph framework. Specifically, we modify two crucial hyperparameters that significantly impact the scale of learnable parameters: the number of graph transformer layers L', and the hidden dimensionality d. We assess the model's performance on the link prediction task by measuring Recall@20. Additionally, we evaluate the computational time for 100 training steps and 100 test steps. The results are illustrated in Figure 3. We summarize the key findings as follows:

Number of graph transformer layers. It can be observed that both training and testing time of OpenGraph exhibit a linear increase as the number of graph transformer layers grows. However, the expansion in model size does not consistently lead to performance improvement. The lack of performance enhancement can be attributed to the overfitting effect and the increased training complexity associated with deep transformer networks.



(a) Performance and time change on the OGBL-ddi dataset.



(b) Performance and time change on the ML-1M dataset.

Figure 3: The impact of model scale on downstream performance and training/testing time (seconds).

Hidden dimensionality. In contrast to the number of transformer layers, the hidden dimensionality leads to quadratic growth in computational time. This reflects the rapid increase in model capacity required to accommodate more complex structural data. Consequently, we observe significant improvements in model performance, surpassing the performance curve for the graph transformer layers. Despite the increased model capacity, the growth in hidden dimensionality facilitates the $\mathbb{R}^N \to \mathbb{R}^d$ projection, reducing information loss and enhancing the quality of the graph token sequence for the subsequent graph transformer layers.

4.7 End-to-End Training Performance (RQ6)

To assess the modeling capabilities of our OpenGraph framework, we perform a performance comparison between OpenGraph and the baselines trained on the same few-shot datasets. Due to space limitations, we present the detailed results and analysis in Appendix A.3.6. The results demonstrate strong graph learning capabilities of our OpenGraph, even in the supervised learning settting.

5 Related Work

Graph Neural Networks have gained attention for their ability to model complex relations in graphs (Wu et al., 2020; Chen et al., 2020). GNNs use message passing to propagate information from neighboring nodes (Jin et al., 2021; Yuan et al., 2020). Representative methods include Graph Convolutional Networks (GCNs) (Gao et al., 2018; Zhang et al., 2021) and Graph Attention Networks (GATs) (Zhang et al., 2022b; Liao et al., 2019). OpenGraph is inspired by the Graph Transformer (Yun et al., 2019; Hu et al., 2020), known for capturing global dependencies in graphs.

Self-Supervised Graph Learning aims to address the limited labeled data issue in graph tasks. These methods leverage graph structure and patterns for data-efficient training (Wu et al., 2021b; Lee et al., 2022; Xia et al., 2023; Xiao et al., 2022; Yang et al., 2024). Graph contrastive learning frameworks, such as GraphCL (You et al., 2020) and SGL (Wu et al., 2021a), create meaningful representations by contrasting positive and negative samples using stochastic data augmenters. Adaptive augmentation schemes like JOAO (You et al., 2021) and GCA (Zhu et al., 2021) have been proposed. DGCL (Li et al., 2021) and UMGRL (Mo et al., 2023) address disentangling factors in contrastive learning. However, these solutions struggle with generalization. OpenGraph enhances graph model generalization across different tasks.

LLM-based Graph Analysis. Recent advancements in LLMs have prompted interest in utilizing them for enhanced graph comprehension and analysis (Ren et al., 2024). GraphLLM (Chai et al., 2023) and GraphQA (Fatemi et al., 2023) transform graphs into natural language descriptions, enabling improved interpretation and reasoning with LLMs. Techniques like instruction tuning in GraphEdit (Guo et al., 2024) and GraphGPT (Tang et al., 2024) incorporate rich textual information from text-attributed graphs for fine-tuning LLMs. However, in certain domains like user behavior graphs and neuronal graphs, obtaining high-quality textual features associated with graph nodes can be challenging. Therefore, there is a need for a graph model that can capture universal structural patterns from graphs, even in the absence of textual data.

6 Conclusion

This research aims to develop an adaptable framework for capturing complex topological patterns in diverse graph structures. Our model demonstrates exceptional generalization capabilities in zero-shot graph learning tasks across various applications. We utilize a scalable graph transformer architecture and LLM-enhanced data augmentation for efficiency and robustness. Extensive experiments on benchmark datasets validate our model's performance. Future plans include incorporating counterfactual learning to discover noisy connections and influential structures while learning universal and transferable structural patterns in diverse graphs.

7 Limitations

Our study serves as an initial exploration of graph foundation models, focusing on distilling the generalization capabilities from LLMs without relying on textual features. However, it is crucial to acknowledge and address the limitations that require further attention in future studies.

Firstly, it is important to note that OpenGraph currently does not include modeling for heterogeneous relations and node types. This limitation may impact its performance and generalization capabilities, particularly when dealing with graph data that exhibits strong heterogeneity, such as knowledge graphs. Future research should prioritize the incorporation of heterogeneous representation learning modules to enable a wider range of applications.

Secondly, graphs are highly versatile data structures that can be applied to a wide range of domains. While we have evaluated the performance of OpenGraph on 8 datasets spanning different domains, it is crucial to further strengthen the experimental validation by testing it on additional datasets from even more diverse application domains. This will provide a more comprehensive understanding of its effectiveness and applicability.

Lastly, while OpenGraph shows promising generalization capability, its explainability remains unexplored. This not only hinders its applicability due to its black box nature, but also hinders us from gaining a deeper understanding of the underlying principles that drive its strong generalization capabilities. Future research should prioritize investigating techniques and methodologies to enhance the explainability of OpenGraph, allowing researchers and practitioners to gain insights into the internal workings of the model and to ensure its reliable and transparent deployment in real-world applications.

In summary, future studies should focus on incorporating heterogeneous representation learning, expanding the range of tested datasets, and enhancing the explainability of OpenGraph for broader applicability and deeper insights.

References

Ziwei Chai, Tianjie Zhang, Liang Wu, et al. 2023. Graphllm: Boosting graph reasoning ability of large language model. *arXiv preprint arXiv:2310.05845*.

Hao Chen, Zefan Wang, Feiran Huang, et al. 2022a. Generative adversarial framework for cold-start item recommendation. In *SIGIR*, pages 2565–2571.

Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. 2022b. Graph unlearning. In *SIGSAC*, pages 499–513.

Ming Chen, Zhewei Wei, Zengfeng Huang, et al. 2020. Simple and deep graph convolutional networks. In *ICML*, pages 1725–1735. PMLR.

Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. 2023. Universal prompt tuning for graph neural networks. *NeurIPS*.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. In *NeurIPS 2023 Workshop*.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *KDD*, pages 1416–1424.

Alan E Gelfand. 2000. Gibbs sampling. *Journal of the American statistical Association*, 95(452):1300–1304.

Shurui Gui, Xiner Li, Limei Wang, and Shuiwang Ji. 2022. Good: A graph out-of-distribution benchmark. *NeurIPS*, 35:2059–2073.

Zirui Guo, Lianghao Xia, Yanhua Yu, et al. 2024. Graphedit: Large language models for graph structure learning. *arXiv preprint arXiv:2402.15183*.

Xiangnan He, Kuan Deng, et al. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*, pages 639–648.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *WWW*, pages 2704–2710.

Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *Recsys*, pages 135–142.

Di Jin, Zhizhi Yu, Cuiying Huo, Rui Wang, Xiao Wang, Dongxiao He, and Jiawei Han. 2021. Universal graph convolutional networks. *NeurIPS*, 34:10654–10664.

Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. 2022. Automated self-supervised learning for graphs. In *ICLR*.

Wei Jin, Yao Ma, Xiaorui Liu, et al. 2020. Graph structure learning for robust graph neural networks. In *KDD*, pages 66–74.

Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks.

Namkyeong Lee, Junseok Lee, and Chanyoung Park. 2022. Augmentation-free self-supervised learning on graphs. In *AAAI*, volume 36, pages 7372–7380.

Haoyang Li, Xin Wang, Ziwei Zhang, et al. 2021. Disentangled contrastive learning on graphs. *NeurIPS*, 34:21872–21884.

Renjie Liao, Yujia Li, Yang Song, et al. 2019. Efficient graph generation with graph recurrent attention networks. *NeurIPS*, 32.

Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, et al. 2022. Graph self-supervised learning: A survey. *TKDE*, 35(6):5879–5900.

Zemin Liu, Xingtong Yu, et al. 2023. Graphprompt: Unifying pre-training and downstream tasks for graph neural networks. In *WWW*, pages 417–428.

Qingsong Lv, Ming Ding, Qiang Liu, et al. 2021. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *KDD*, pages 1150–1160.

Yujie Mo, Yajie Lei, Jialie Shen, et al. 2023. Disentangled multiplex graph representation learning. In *ICML*, pages 24983–25005. PMLR.

Xubin Ren, Wei Wei, Lianghao Xia, et al. 2024. Representation learning with large language models for recommendation. *WWW*.

Aravind Sankar, Yozen Liu, Jun Yu, et al. 2021. Graph neural networks for friend ranking in large-scale social platforms. In *WWW*, pages 2535–2546.

Mingchen Sun, Kaixiong Zhou, et al. 2022. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *KDD*, pages 1717–1727.

Xiangguo Sun, Hong Cheng, Jia Li, et al. 2023. All in one: Multi-task prompting for graph neural networks. In *KDD*.

Jiabin Tang, Yuhao Yang, Wei Wei, et al. 2024. Graphgpt: Graph instruction tuning for large language models. *SIGIR*.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018a. Graph attention networks.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018b. Deep graph infomax. In *ICLR*.

Xiaoyang Wang, Yao Ma, Yiqi Wang, et al. 2020. Traffic flow prediction via spatial temporal graph neural network. In *WWW*, pages 1082–1092.

Wei Wei, Jiabin Tang, Yangqin Jiang, et al. 2024. Promptmm: Multi-modal knowledge distillation for recommendation with prompt-tuning. In *WWW*.

Jiancan Wu, Xiang Wang, Fuli Feng, et al. 2021a. Self-supervised graph learning for recommendation. In *SIGIR*, pages 726–735.

Lirong Wu, Haitao Lin, Cheng Tan, et al. 2021b. Self-supervised learning on graphs: Contrastive, generative, or predictive. *TKDE*.

Qitian Wu, Hengrui Zhang, Junchi Yan, and David Wipf. 2021c. Handling distribution shifts on graphs: An invariance perspective. In *ICLR*.

Zonghan Wu, Shirui Pan, Fengwen Chen, et al. 2020. A comprehensive survey on graph neural networks. *TNNLS*, 32(1):4–24.

Lianghao Xia, Chao Huang, Chunzhen Huang, et al. 2023. Automated self-supervised learning for recommendation. In *WWW*, pages 992–1002.

Teng Xiao, Zhengyu Chen, Zhimeng Guo, et al. 2022. Decoupled self-supervised learning for graphs. *NeurIPS*, 35:620–634.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? In *ICLR*.

Yuhao Yang, Lianghao Xia, Da Luo, Kangyi Lin, and Chao Huang. 2024. Graph pre-training and prompt learning for recommendation. In *WWW*.

Rex Ying, Ruining He, Kaifeng Chen, et al. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pages 974–983.

Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph contrastive learning automated. In *ICML*, pages 12121–12132. PMLR.

Yuning You, Tianlong Chen, Yongduo Sui, et al. 2020. Graph contrastive learning with augmentations. *NeurIPS*, 33:5812–5823.

Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. Xgnn: Towards model-level explanations of graph neural networks. In *KDD*, pages 430–438.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *NeurIPS*, 32.

Qiannan Zhang, Xiaodong Wu, et al. 2022a. Few-shot heterogeneous graph learning via cross-domain knowledge transfer. In *KDD*, pages 2450–2460.

Wentao Zhang, Ziqi Yin, Zeang Sheng, et al. 2022b. Graph attention multi-layer perceptron. In *KDD*, pages 4560–4570.

Yiding Zhang, Xiao Wang, Chuan Shi, Nian Liu, and Guojie Song. 2021. Lorentzian graph convolutional networks. In *WWW*, pages 1249–1261.

Elena Zheleva and Lise Getoor. 2007. Preserving the privacy of sensitive relationships in graph data. In *KDD workshop*, pages 153–171. Springer.

Yanping Zheng, Hanzhi Wang, Zhewei Wei, Jiajun Liu, and Sibo Wang. 2022. Instant graph neural networks for dynamic graphs. In *KDD*, pages 2605–2615.

Yanqiao Zhu, Yichen Xu, Feng Yu, et al. 2021. Graph contrastive learning with adaptive augmentation. In *WWW*, pages 2069–2080.

A Appendix

The appendix offers additional details about our OpenGraph framework. It covers the neural model component, the generation algorithm, as well as the experimental settings and supplementary results.

A.1 Methodology

A.1.1 Zero-Shot Node Classification

Node classification tasks face challenges when transferring trained classification capabilities from one graph to another due to the heterogeneity of node classes across datasets. To address this challenge, our OpenGraph transforms the graph-specific classification task into a unified link prediction task. This task involves predicting links between regular nodes and special nodes representing different classes. By leveraging the generalized topology extraction capability and learning from observed class-node relations, our OpenGraph enables zero-shot node classification.

A.1.2 Handling Node Features

In attributed graphs, node attributes can vary across different graphs, including textual, numerical, and categorical features. To address the semantic differences in node features, our OpenGraph transforms these attributes into a unified graph structure format. This format can be easily tokenized by our unified graph tokenizer and comprehended by our trained graph transformer. In our approach, we sample node pairs with the highest similarity scores as augmented edges. The similarity scores $s_{i,j}$ between a node pair (v_i, v_j) are calculated as $s_{i,j} = \mathbf{f}_i^\top \mathbf{f}_j$. Using these similarity scores, we select the top $B \times K$ edges for each batch, where each batch contains $B \times |\mathcal{V}|$ candidate edges. This strategy is applied to all experimental datasets that have node features.

A.1.3 Details of Scalable Graph Transformer

In the efficient self-attention with anchors, we determine the number of anchors S by $S = d/H < B$, to ensure that the self-attention module incurs similar memory costs as other fully-connected components. Here, H represents the number of attention head. More specifically, the self-attention process for each head can be summarized as follows:

$$\mathbf{e}_t^{(3)} = \sum_{v_a} \alpha_{t,a} \mathbf{W}^{(v)} \mathbf{e}_a^{(2)}, \ \mathbf{e}_a^{(2)} = \sum_{v_t} \alpha_{a,t} \mathbf{W}^{(v)} \mathbf{e}_t^{(1)}$$

$$\alpha_{t,a} = \text{softmax}\Big(\frac{(\mathbf{W}^{(q)}\mathbf{e}_t)^\top \cdot (\mathbf{W}^{(k)}\mathbf{e}_a)}{\sqrt{d/H}}\Big) \qquad (7)$$

Our efficient self-attention involves embeddings $\mathbf{e}_*^{(1)}, \mathbf{e}_*^{(2)}, \mathbf{e}_*^{(3)}$ for anchor nodes v_a and vanilla nodes v_t. After each attention calculation, the results from multiple heads are concatenated, passed through a learnable linear layer, and connected with a residual connection. The parameters $\mathbf{W}^{(q)}, \mathbf{W}^{(k)}, \mathbf{W}^{(v)}$ are the parameters of the attention layer. To reduce computational complexity, we employ a two-stage self-attention process. It transforms the $3 \times B$-length sequence to a shorter S-length sequence and then reverses the process.

After the self-attention module, each layer of our scalable graph transformer includes a two-layer fully-connected block with residual connections, accompanied by two layer normalization modules. To ensure numerical stability, per-layer scaling is applied by element-wisely dividing embeddings by a selected constant $K = 10$.

A.1.4 Model Optimization

To optimize our OpenGraph model, we utilize the masked autoencoding (MAE) training paradigm for self-supervised pre-training. Let's denote our OpenGraph as f, with trainable parameters $\mathbf{\Theta}_f$, and a graph projection function ϕ. The model is trained on a set of graphs $\mathcal{G}_s$ with batch-specific labels $\bar{\mathcal{E}}_s$. The objective of the generative SSL optimization is defined as follows:

$$\arg\min_{\mathbf{\Theta}_f} \sum_{\mathcal{G}_s} \sum_{\bar{\mathcal{E}}_s \in \mathcal{G}_s} \mathcal{L}\Big(f(\mathcal{G}_s - \bar{\mathcal{E}}_s, \phi; \mathbf{\Theta}_f), \bar{\mathcal{E}}_s\Big) + \lambda \cdot \|\mathbf{\Theta}_f\|_\mathrm{F}^2 \qquad (8)$$

Here, $\mathcal{G}_s - \bar{\mathcal{E}}_s$ represents the input graph $\mathcal{G}_s$ with the label edge set $\bar{\mathcal{E}}$ removed. All training graphs $\mathcal{G}_s$ are jointly trained with random alternations. To enhance the model's adaptability to different graph projections ϕ, we regenerate the projection function ϕ for each training graph $\mathcal{G}_s$ every 10 training steps. λ denotes the weight for L_2 regularization.

A.2 Graph Generation Algorithm

A.2.1 Prompt Template and Examples of Generated Nodes

In this section, we present our prompt strategy for leveraging the Language Model (LLM) to divide general nodes into more fine-grained entities. Figure 4 illustrates our prompt template, with key parameters highlighted in red. We provide concrete examples of prompt parameters and showcase the generation results for both the e-commerce scenario and the venue rating scenario.

Prompt Template
List all distinct sub-categories of **{entity_name}** within the **{prefix}** category in the context of **{scenario_desc}**, ensuring a finer level of granularity. The sub-categories should not overlap with each other. And a sub-category should be a smaller subset of **{entity_name}**. Directly present the list EXACTLY following the form: "sub-category a, sub-category b, sub-category c, ..." without other words, format symbols, new lines, serial numbers.

Prompt Example
entity_name = “women’s clothing”
scenario_desc = “e-commerce platform like Amazon”
prefix = “products, clothing”

Examples of Generated Nodes
products, Clothing, Women's clothing, Sweaters, Crewneck sweaters
products, Clothing, Men's clothing, Costumes, Scary costumes
products, Clothing, Outerwear, Vests, Sweater Vests
products, Shoes, Flats, T-strap flats, Open toe T-strap flats
products, Shoes, Ballet flats, Ankle strap ballet flats, Nude ankle strap ballet flats
products, Jewelry, Jewelry Sets, Choker, Gothic Choker
products, Electronics, office electronics, Calculators, Scientific Calculators
products, Books, Non-fiction, Self-Help, Codependency

Prompt Example
entity_name = “Restaurant”
scenario_desc = “venue rating platform like Yelp”
prefix = “business venues”

Examples of Generated Nodes
business venues, Restaurant, American, Barbecue, BBQ fusion
business venues, Restaurant, Buffet, Chinese buffet, Seafood
business venues, Cafe, Tea house, Tea room, British tea house
business venues, Cafe, Brunch spot, Buffet brunch, Vegan buffet
business venues, Bar, Karaoke Bar, Karaoke DJ nights, live band karaoke
business venues, Nightclub, Live Music Venue, Jazz Club, Latin Jazz Club
business venues, Fast Food Restaurant, Smoothie Bar, Specialty smoothie bar, Fresh fruit smoothie bar
business venues, Drive-Thru Restaurant, Fast food, Pizza place, Coal-fired pizza

Figure 4: Prompt template and generation examples.

A.2.2 Node Generation Algorithm

We elaborate the process of our tree-of-prompt algorithm that traversing the vertex space for a specific application scenario, as in Algorithm 1.

A.2.3 Edge Generation Algorithm

We elaborate our edge generation algorithm based on LLM-given node representations and the Gibbs sampling algorithm in Algorithm 2. Here we illustrate the case for generating person-entity relations, which is more complex compared to the entity-entity relation generation.

A.3 Experiments

A.3.1 Experimental Datasets

Our experimental datasets include 5 link prediction datasets and 3 node classification datasets. The data statistics are summarized in Table 4.

Link prediction datasets. We employ five link prediction datasets from diverse application scenarios. The objective of these datasets is to predict the most likely connections for each node based on previous observations of node-wise interactions.

- **OGBL-ddi**. This dataset is used for drug-drug interaction prediction. Each node represents a drug. The edges represent the combined effect of taking two drugs together, which differs significantly from taking them individually.

Algorithm 1: Tree-of-prompt algorithm for node generation.

```
Input: Name for the initial general node $v_0$ ($e.g.$ 'products'), text descriptions
       for the application scenario $S$ ($e.g.$ 'e-commerce platform like Amazon'),
       maximum depth $D$ of the prompt tree
Output: Generated nodes $\hat{\mathcal{V}}$.
1  Function DivideNode($v$, $n$):
2      if $n \geq D$ then
3          return $[v]$
4      end
5      $\bar{\mathcal{V}} = \text{LLM}(v, S)$
6      $\hat{\mathcal{V}} = []$
7      foreach $v' \in \bar{\mathcal{V}}$ do
8          $\hat{\mathcal{V}} += $ DivideNode($v'$, $n+1$)
9      end
10     return $\hat{\mathcal{V}}$
11 return DivideNode($v_0$, 1)
```

Table 4: Statistics of experimental datasets.

	Dataset	# Node	# Edge	# Feat	# Class
Link	OGBL-ddi	4,267	1,334,889	0	N/A
	OGBL-collab	235,868	1,285,465	128	
	ML-1M	9,746	720,152	0	
	ML-10M	80,555	7,200,040	0	
	Amazon-book	144,242	2,380,730	0	
Node	Cora	2,708	10,556	1433	7
	Citeseer	3,327	9,104	3,703	6
	Pubmed	19,717	88,648	500	3
Gen.	Gen0	46,861	454,276	0	N/A
	Gen1	51,061	268,007	0	
	Gen2	32,739	240,500	0	

- **OGBL-collab**. It is an academic social relation dataset. Its nodes represent scholars, and edges denote collaborations. Each node is combined with a 128-dimensional average word embedding calculated from the author’s publications.
- **Movielens-1M & Movielens-10M**. These two datasets are both collected from the movie rating platform Movielens. The graphs are constructed by connecting users with the movies they have rated. The two datasets contain 1 million and 10 million rating records, respectively.
- **Amazon-book**. This dataset contains review data from the Amazon platform. The nodes in the dataset represent users and books, while the edges denote the review records between them.

Node classification datasets. For the node classification task, we utilize three widely-used citation network datasets: **Cora**, **Citeseer**, and **Pubmed**. In these datasets, each node represents an academic paper, and an edge (v_i, v_j) denotes a citation relation from paper node v_i to v_j. The Cora and Citeseer datasets include binary bag-of-words vectors as node features, while the Pubmed dataset utilizes TF-IDF weighted word vectors as node features. The objective of these datasets is to classify each node into predefined paper categories based on the citation relations and node attributes.

A.3.2 Evaluation Protocol

This section includes detailed description for our cross-dataset zero-shot setting and the few-shot settings for baselines. It also introduces the evaluation metrics used in our experiments.

Zero-shot setting. For our OpenGraph, we utilize a zero-shot learning setting in which OpenGraph is not trained on any of these real-world datasets but is tested using the training set information as input information, including the graph structures, node features, and node labels in the training set. To effectively generalize to unseen node labels in node classification with zero shot, taking inspiration from previous works (Sun et al., 2022), we treat the label classes as new nodes and connect the vanilla nodes with training labels to the corresponding class nodes. This strategy removes the requirement for learning class-related parameters in the zero-shot learning setting. This enhancement is also applied to baselines methods.

Few-shot setting. Since most baselines perform poorly in the foregoing zero-shot setting, we evaluate them in the one-shot setting and five-shot setting. In the node classification task, the *k*-shot setting refers to preserving a maximum of *k* training instances for each label class. For the link prediction task, the *k*-shot training set contains at most *k* links for each node. Non-pretraining approaches such as MLP and GNNs are solely trained on the few-shot training set. On the other hand, baselines following the pretraining-and-tuning paradigm undergo pretraining and subsequent tuning on the few-shot set. In link prediction, they are pretrained on the same generated datasets as our OpenGraph. Model parameters that are not transferable across datasets are re-learned during the tuning phase. In node classification, these methods are pretrained on the graph of the target dataset and fine-tuned on the classification labels. In the test phase, all information in the training set is employed.

Evaluation metrics. In link prediction, we follow existing works (Wei et al., 2024) to conduct the full-rank test for each node. To be specific, for each node, all nodes not connected to it in the training set are ranked by the model. The top-*N* nodes are taken as positive predictions, and we calculate *Recall@N* scores with $N = 20, 40$. In node classification, we employ the widely-used *Accuracy* and *Macro-F1* metrics (Chen et al., 2022b).

A.3.3 Implementation Details

We implemented our OpenGraph framework using PyTorch. The model employs the Adam optimizer with a learning rate of $1e-4$ or $5e-5$. The learnable parameters are initialized using the Xavier uniform initialization method. By default, the reported performance is achieved by OpenGraph with an embedding size of $d = 1024$ and a maximum power order of $L = 3$ for adjacency smoothing. The default scalable graph transformer utilizes $L' = 3$ transformer layers, $H = 4$ attention heads, and $S = 256$ sampled anchor nodes. The training batch size, which is also used for token sequence sampling, is set as $B = 1024$.

The reported results are obtained by pretraining our OpenGraph network using three generated datasets: Gen0, Gen1, and Gen2. The statistics of these datasets are presented in Table 4. We first generate the Gen0 dataset without injecting the graph's topological pattern. Subsequently, we generate Gen1 and Gen2 based on Gen0 by incorporating the graph pattern. In comparison to Gen1, the Gen2 dataset undergoes an additional densification process, where nodes with less than 10 edges are removed. To acquire the nodes for the Gen0 dataset, we prompt the LLM to iterate through all products on an e-commerce platform, with a maximum generation depth of 5. The Gibbs sampling algorithm is initialized with nodes having 6 random edges. To ensure low overlap between consecutive samples, we introduce a separation of 1000 sampling steps before generating each new sample. The dynamic probability normalization maintains the last $T' = 5000$ sampling instances. The node locality incorporation involves using 7 locality indices and 0.95 decay rate.

The baseline methods are evaluated using their original code, or we closely follow the original code to implement them. Our implementations of the baselines are carefully aligned with the reported performance in their original evaluation settings. We employ grid search to optimize the hyperparameter settings for each baseline.

A.3.4 Baselines

We give detailed descriptions for the baseline models in this section. 9 models from 4 different research lines are utilized in our evaluation.

Graph-agnostic Approaches.

- **MF**. This is the matrix factorization approach which learns node embeddings to reconstruct the observed adjacency matrix. For the node classification task, we adapt it to learn embedding vectors for each node to predict node labels.
- **MLP**. This baseline utilizes a multi-layer perceptron to extract deep features individually for each node. For datasets without node attributes, this baseline learns initial node embeddings.

Non-pretraining Graph Neural Networks.

- **GCN** (Kipf and Welling, 2017). This approach utilizes iterative graph convolutional operators to extract the high-order topological information.
- **GAT** (Veličković et al., 2018a). This graph attention network learns weights for node-wise connections using the attention mechanism, to facilitate adaptive graph information propagation.
- **GIN** (Xu et al., 2018). This method enhances the representation power of GNNs by employing a distinct graph encoding method that emphasizes the discrimination of non-isomorphic structures.

Graph Pre-training Models.

- **GraphCL** (Zhu et al., 2021). This baseline method utilizes pre-training of graph models through the application of a self-discriminative contrastive learning task on learned node embeddings. It incorporates various graph augmentation techniques such as node drop, edge permutation, random walk, and feature masking.
- **DGI** (Veličković et al., 2018b). This method introduces a self-supervised pre-training task that aims to maximize the mutual information between the local view and the global view.

Graph Prompt Tuning Methods.

- **GraphPrompt** (Liu et al., 2023). This work presents a unified framework for pre-training and prompt tuning of graph models. It introduces a learnable prompt layer that automatically identifies crucial information in the pre-trained model to facilitate downstream tasks.
- **GPF** (Fang et al., 2023). This is a universal graph prompt tuning framework designed for various graph pre-training strategies. It introduces two versions of a learnable graph prompt layer.

Table 5: Performance comparison with models trained on few-shot datasets, in terms of Recall@20 (%).

Model	GCN		GAT		GIN		OpenGraph	
shot	1	5	1	5	1	5	1	5
ddi	2.79	7.05	5.80	7.11	5.30	7.35	**7.93**	**8.60**
ML1M	4.32	10.54	2.45	15.06	4.66	14.58	**16.58**	**19.19**
Amazon	0.96	2.51	0.47	2.28	0.69	2.52	**2.96**	**3.10**

A.3.5 Alternative Projection Methods (RQ2)

- **One-hot encoding**. This graph projection strategy employs a large table of low-dimensional embeddings for node ids, where nodes with the same index from different datasets are directly mapped to the same embedding vector. Specifically, we utilize 100,000 independent embedding vectors. For datasets with more nodes, we use the remainder of 100,000 dividing node indices.
- **Degree embeddings**. Using degree embeddings for node representation is a commonly used strategy for non-attributed graphs. Each degree number is assigned an independent learnable embedding vector, and each node is initially represented by its degree representation.
- **Random projection**. In this approach, a random representation vector is assigned to each node, sampled from a uniform distribution. With a sufficiently large representation space, this method aims to approximately distribute nodes with equal distances from one another. As a result, this projection method does not rely on any specific assumptions about the node distribution. This characteristic allows it to outperform the other two strategies, which are based on certain assumptions and are thus more prone to overfitting the pre-training dataset.

A.3.6 End-to-end Training (RQ6)

We compare our OpenGraph with other graph encoding methods in the supervised learning setting. The models are trained using the 1-shot and 5-shot training sets from OGBL-ddi, ML-1M, and Amazon-book, and then tested on the corresponding test set. Without pre-training, this experiment aims to examine the modeling capacity for different graph neural architectures. From the results shown in Table 5, we draw the following conclusions:

- **Superior modeling capabilities of OpenGraph**. Our OpenGraph achieves best performance on all tested datasets, demonstrating the superior graph learning ability for OpenGraph. We attribute this superiority to the precise preservation of structural information by our graph tokenization module, and the strength of our scalable graph transformer in learning global relations.

- **Robustness of OpenGraph**. We notice that our OpenGraph exhibits less performance degradation on the more sparse 1-shot datasets. This demonstrates the inherent robustness of OpenGraph's model architecture. Such robustness can be ascribed to the effectiveness of the fast topology projection, which effectively captures key graph structures even without sufficient training.

Algorithm 2: Edge generation algorithm.

```
Input: Node embedding table $\mathbf{H}$ given by the LLM, node set $\mathcal{V}$, maximum
       locality index $N$, locality decay factor $\alpha$, dynamic probability
       normalization range $T'$, number of sampling steps to draw a new
       sample $T_0$, number of initial sampling steps to skip for data
       quality $T_1$, number of sampling steps to shift current locality index
       $T_2$, maximum sampling steps $T_{\text{max}}$.
Output: List of interactions $\mathcal{I}$.
1  Draw a random interaction sample $\mathbf{a}^0$
2  Initialize current locality index $n = 0$
3  Initialize the pool for dynamic probability $\mathcal{P} = []$
4  Initialize $\mathcal{I} = []$ for $t = 1$ to $T_{max}$ do
5      if $t$ mod $T_2 == 0$ then
6          $n = (n + 1) \bmod N$
7      end
8      $i = t \bmod |\mathcal{V}|$
9      $p = \sum_{v_i} a_i^t (\mathbf{h}_i / \|\mathbf{a}^t\|_0)^\top \cdot \mathbf{h}_{t'}$
10     $\mathcal{P} += [p]$
11     if $|\mathcal{P}| > T'$ then
12         $\mathcal{P} = \mathcal{P}[-T' :]$
13     end
14     $\mu = \text{mean}(\mathcal{P})$, $\sigma = \text{std}(\mathcal{P})$
15     $\bar{p} = (p - \mu)/(4\sigma)$
16     $\hat{p} = \bar{p} \cdot \alpha^{|n - n_i|}$
17     Decide if $\mathbf{a}^t \oplus v_{t'}$ is accepted accordding to $\hat{p}$
18     if $t \geq T_1$ and $t$ mod $T_0 == 0$ then
19         $\mathcal{I} += [\mathbf{a}^t \oplus v_{t'}]$
20     end
21 end
22 return $\mathcal{I}$
```