GPT-GNN: Generative Pre-Training of Graph Neural Networks

Ziniu Hu
University of California, Los Angeles
bull@cs.ucla.edu

Yuxiao Dong
Microsoft Research, Redmond
yuxdong@microsoft.com

Kuansan Wang
Microsoft Research, Redmond
kuansanw@microsoft.com

Kai-Wei Chang
University of California, Los Angeles
kwchang@cs.ucla.edu

Yizhou Sun
University of California, Los Angeles
yzsun@cs.ucla.edu

ABSTRACT

Graph neural networks (GNNs) have been demonstrated to be powerful in modeling graph-structured data. However, training GNNs usually requires abundant task-specific labeled data, which is often arduously expensive to obtain. One effective way to reduce the labeling effort is to pre-train an expressive GNN model on unlabeled data with self-supervision and then transfer the learned model to downstream tasks with only a few labels. In this paper, we present the GPT-GNN* framework to initialize GNNs by generative pre-training. GPT-GNN introduces a self-supervised attributed graph generation task to pre-train a GNN so that it can capture the structural and semantic properties of the graph. We factorize the likelihood of the graph generation into two components: 1) Attribute Generation and 2) Edge Generation. By modeling both components, GPT-GNN captures the inherent dependency between node attributes and graph structure during the generative process. Comprehensive experiments on the billion-scale Open Academic Graph and Amazon recommendation data demonstrate that GPT-GNN significantly outperforms state-of-the-art GNN models without pre-training by up to 9.1% across various downstream tasks.

CCS CONCEPTS

• **Computing methodologies → Unsupervised learning**; **Neural networks**; **Learning latent representations**.

KEYWORDS

Generative Pre-Training; Graph Neural Networks; Graph Representation Learning; Network Embedding; GNN Pre-Training

ACM Reference Format:
Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403237

1 INTRODUCTION

The breakthroughs in graph neural networks (GNNs) have revolutionized graph mining from structural feature engineering to representation learning [1, 9, 17]. Recent GNN developments have been demonstrated to benefit various graph applications and network tasks, such as semi-supervised node classification [17], recommendation systems [42], and knowledge graph inference [28].

Commonly, GNNs take a graph with attributes as input and apply convolutional filters to generate node-level representations layer by layer. Often, a GNN model is trained with supervised information in an end-to-end manner for one task on the input graph. That said, for different tasks on the same graph, it is required to have enough and different sets of labeled data to train dedicated GNNs corresponding to each task. Usually, it is arduously expensive and sometimes infeasible to access sufficient labeled data for those tasks, particularly for large-scale graphs. Take, for example, the author disambiguation task in academic graphs [34], it has still faced the challenge of the lack of ground-truth to date.

Similar issues had also been experienced in natural language processing (NLP). Recent advances in NLP address them by training a model from a large unlabeled corpus and transferring the learned model to downstream tasks with only a few labels—the idea of pre-training. For example, the pre-trained BERT language model [4] is able to learn expressive contextualized word representations by reconstructing the input text—next sentence and masked language predictions, and thus it can significantly improve the performance of various downstream tasks. Additionally, similar observations have also been demonstrated in computer vision [2, 13, 35].

Inspired by these developments, we propose to pre-train graph neural networks for graph mining. The goal of the pre-training is to empower GNNs to capture the structural and semantic properties of a input graph, so that it can easily generalize to any downstream tasks with a few fine-tuning steps on the graphs within the same domain. To achieve this goal, we propose to model the graph distribution by learning to reconstruct the input attributed graph.

To pre-train GNNs based on graph reconstruction, one straightforward option could be to directly adopt the neural graph generation techniques [16, 18, 43]. However, they are not suitable for pre-training GNNs by design. First, most of them focus only on generating graph structure without attributes, which does not capture the underlying patterns between node attributes and graph structure—the core of convolutional aggregation in GNNs. Second, they are designed to handle small graphs to date, limiting their potential to pre-train on large-scale graphs.

[*] The code and pre-trained models are available at https://github.com/acbull/GPT-GNN.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403237

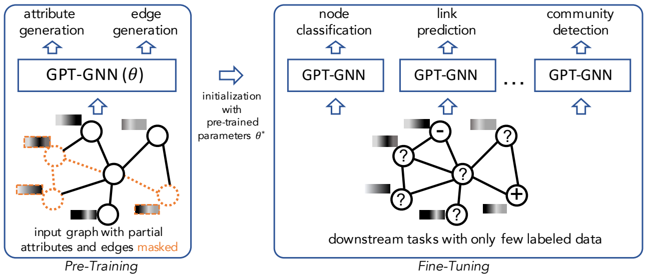


Figure 1: The pre-training and fine-tuning flow of GPT-GNN: First, a GNN is pre-trained with the self-supervised learning task—attribute and structure generations. Second, the pre-trained model and its parameters are then used to initialize models for downstream tasks on the input graph or graphs of the same domain.

Contributions. In this work, we design a self-supervised attributed graph generation task for GNN pre-training, with which both the structure and attributes of the graph are modeled. Based on this task, we present the GPT-GNN framework for generative pre-training of graph neural networks (Cf. Figure 1). The pre-trained GNN on the input graph can be then used as the initialization of models for different downstream tasks on the same type of graphs. Specifically, our contributions are illustrated below.

First, we design an attributed graph generation task to model both node attributes and graph structure. We decompose the graph generation objective into two components: Attribute Generation and Edge Generation, whose joint optimization is equivalent to maximizing the probability likelihood of the whole attributed graph. In doing this, the pre-trained model can capture the inherent dependency between node attributes and graph structure.

Second, we propose an efficient framework GPT-GNN to conduct generative pre-training with the aforementioned task. GPT-GNN can calculate the attribute and edge generation losses of each node simultaneously, and thus only need to run the GNN once for the graph. Additionally, GPT-GNN can handle large-scale graphs with sub-graph sampling and mitigate the inaccurate loss brought by negative sampling with an adaptive embedding queue.

Finally, we pre-train GNNs on two large-scale graphs—the Open Academic Graph (OAG) of 179 million nodes & 2 billion edges and Amazon recommendation data of 113 million nodes. Extensive experiments show that the GPT-GNN pre-training framework can significantly benefit various downstream tasks. For example, by applying the pre-trained model on OAG, the node classification and link prediction performance is on average lifted by 9.1% over the state-of-the-art GNN models without pre-training. In addition, we show that GPT-GNN can consistently improve the performance of different base GNNs under various settings.

2 PRELIMINARIES AND RELATED WORK

The goal of pre-training is to allow a model (usually neural networks) to initialize its parameters with pre-trained weights. In this way, the model can leverage the commonality between the pre-training and downstream tasks. Recently pre-training has shown superiority in boosting the performance of many downstream applications in computer vision and natural language processing. In the following, we first introduce the preliminaries about GNNs and then review pre-training approaches in graphs and other domains.

2.1 Preliminaries of Graph Neural Networks

Recent years have witnessed the success of GNNs for modeling graph data [12, 15, 17, 36]. A GNN can be regarded as using the input graph structure as the computation graph for message passing [9], during which the local neighborhood information is aggregated to get a more contextual representation. Formally, suppose $H_t^{(l)}$ is the node representation of node t at the (l)-th GNN layer, the update procedure from the $(l$-$1)$-th layer to the (l)-th layer is:

$$H_t^{(l)} \leftarrow \underset{\forall s\in N(t), \forall e\in E(s,t)}{\textbf{Aggregate}} \left(\left\{ \textbf{Extract}\left(H_s^{(l-1)}; H_t^{(l-1)}, e\right) \right\} \right), \qquad (1)$$

where $N(t)$ denotes all the source nodes of node t and $E(s,t)$ denotes all the edges from node s to t.

There are two basic operators for GNNs, which are **Extract(·)** and **Aggregate(·)**. Among them, **Extract(·)** represents the neighbor information extractor. It uses the target node's representation $H_t^{(l-1)}$ and the edge e between the two nodes as query, and extract useful information from source node $H_s^{(l-1)}$. **Aggregate(·)** serves as the aggregation function of the neighborhood information. The *mean, sum,* and *max* functions are often considered as the basic aggregation operators, while sophisticated pooling and normalization functions can also be designed. Under this framework, various GNN architectures have been proposed. For example, the graph convolutional network (GCN) proposed by Kipf *et al.* [17] averages the one-hop neighbor of each node in the graph, followed by a linear projection and then a non-linear activation. Hamilton *et al.* [12] propose GraphSAGE that generalizes GCN's aggregation operation from *average* to *sum, max* and a *RNN unit*.

Also, there are a bunch of works incorporating the attention mechanism into GNNs. In general, the attention-based models implement the **Extract(·)** operation by estimating the importance of each source node, based on which a weighted aggregation is applied. For example, Velickovi *et al.* [36] propose the graph attention network (GAT), which adopts an additive mechanism to calculate attention and uses the same weight for calculating messages. Recently, Hu *et al.* propose the heterogeneous graph transformer (HGT) [15] that leverages multi-head attentions for different relation types to get type-dependent attentions. The proposed pre-training framework GPT-GNN can apply to all of these GNN models.

2.2 Pre-Training for Graphs

Previous studies have proposed to utilize pre-training to learn node representations, which largely belong to two categories. The first category is usually termed as network/graph embedding, which directly parameterizes the node embedding vectors and optimizes them by preserving some similarity measures, such as the network proximity [33] or statistics derived from random walks [6, 11, 26]. However, the embeddings learned in this way cannot be used to initialize other models for fine-tuning over other tasks. In contrast, we consider a transfer learning setting, where the goal is to pre-train a generic GNN that can deal with different tasks.

With the increasing focus on GNNs, researchers have explored the direction of pre-training GNNs on unannotated data. Kipf *et al.* propose Variational Graph Auto-Encoders [16] to reconstruct the graph structure. Hamilton *et al.* propose GraphSAGE [12], which can optimize via an unsupervised loss by using random walk based similarity metric. Velickovic *et al.* introduce Graph Infomax [37], which maximizes the mutual information between node representations obtained from GNNs and a pooled graph representation. Although these methods show enhancements over purely-supervised learning settings, the learning tasks can be achieved by forcing nearby nodes to have similar embeddings, ignoring the rich semantics and higher-order structure of the graph. Our work proposes to pre-train GNNs by the permutated generative objective, which is a harder graph task and thus can guide the model to learn more complex semantics and structure of the input graph.

In addition, there are attempts to pre-train GNNs to extract graph-level representations. Sun *et al.* present InfoGraph [29], which maximizes the mutual information between graph-level representations obtained from GNNs and the representations of sub-structures. Hu *et al.* [14] introduce different strategies to pre-train GNNs at both node and graph levels and show that combining them together can improve the performance on graph classification tasks. Our work is different with them as our goal is to pre-train GNNs over a single (large-scale) graph and conduct the node-level transfer.

2.3 Pre-Training for Vision and Language

Pre-training has been widely used in computer vision (CV) and natural language processing (NLP). In CV, early pre-training techniques [5, 10, 24] mostly follow the paradigm of first pre-training a model on large-scale supervised datasets (such as ImageNet [3]) and then fine-tuning the pre-trained model on downstream tasks [10] or directly extracting the representations as features [5]. Recently, some self-supervised tasks [2, 13, 35] have also been utilized to pre-train vision models. In NLP, Early works have been focused on learning (shallow) word embeddings [22, 25] by leveraging the co-occurrence statistics on the text corpus. More recently, significant progresses have been made on contextualized word embeddings, such as BERT [4], XLNET [41] and GPT [27]. Take BERT as an example, it pre-trains a text encoder with two self-supervised tasks in order to better encode words and their contexts. These pre-training approaches have been shown to yield state-of-the-art performance in a wide range of NLP tasks and thus used as a fundamental component in many NLP systems.

3 GENERATIVE PRE-TRAINING OF GNNS

In this section, we formalize the attributed graph generation task and introduce the generative pre-training framework (GPT-GNN).

3.1 The GNN Pre-Training Problem

The input to GNNs is usually an attributed graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, where $\mathcal{V}$ and $\mathcal{E}$ denote its node and edge sets, and $\mathcal{X}$ represents the node feature matrix. A GNN model learns to output node representations under the supervision of a specific downstream task, such as node classification. Sometimes there exist multiple tasks on a single graph, and most GNNs require sufficient dedicated labeled data for each task. However, it is often challenging to obtain sufficient annotations, in particular for large-scale graphs, hindering the training of a well-generalized GNN. Therefore it is desirable to have a pre-trained GNN model that can generalize with few labels. Conceptually, this model should 1) capture the intrinsic structure and attribute patterns underlying the graph and 2) thus benefit various downstream tasks on this graph.

GNN Pre-Training. Formally, our goal of GNN pre-training concerns the learning of a general GNN model f_θ purely based on single (large-scale) graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ without labeled data such that f_θ is a good initialization for various (unseen) downstream tasks on the same graph or graphs of the same domain. To learn such a general GNN model without labeled data on the graph, a natural question arises here is: *how to design an unsupervised learning task over the graph for pre-training the GNN model?*

3.2 The Generative Pre-Training Framework

Recent advances in self-supervised learning for NLP [4, 41] and CV [2, 13, 35] have shown that unlabeled data itself contains rich semantic knowledge, and thus a model that can capture the data distribution is able to transfer onto various downstream tasks. Inspired by this, we propose GPT-GNN, which pre-trains a GNN by reconstructing/generating the input graph's structure and attributes.

Formally, given an input graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ and a GNN model f_θ, we model the likelihood over this graph by this GNN as $p(G; \theta)$—representing how the nodes in G are attributed and connected. GPT-GNN aims to pre-train the GNN model by maximizing the graph likelihood, i.e., $\theta^* = \max_\theta p(G; \theta)$.

Then, the first question becomes how to properly model $p(G; \theta)$. Note that most existing graph generation methods [18, 43] follow the auto-regressive manner to factorize the probability objective, i.e., the nodes in the graph come in an order, and the edges are generated by connecting each new arriving node to existing nodes. Similarly, we denote a permutation vector π to determine the node ordering, where i^π denotes the node id of i-th position in permutation π. Consequently, the graph distribution $p(G; \theta)$ is equivalent to the expected likelihood over all possible permutations, i.e.,

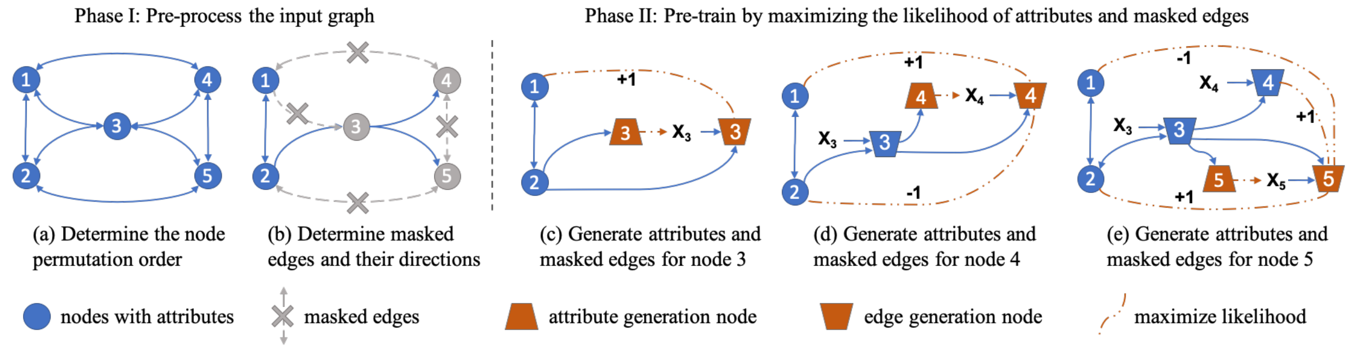
$$p(G; \theta) = \mathbb{E}_\pi \left[p_\theta(X^\pi, E^\pi) \right],$$

where $X^\pi \in \mathbb{R}^{|\mathcal{V}| \times d}$ denotes permutated node attributes and E is a set of edges, while E_i^π denotes all edges connected with node i^π. For simplicity, we assume that observing any node ordering π has an equal probability and also omit the subscript π when illustrating the generative process for one permutation in the following sections. Given a permutated order, we can factorize the log likelihood autoregressively—generating one node per iteration—as:

$$\log p_\theta(X, E) = \sum_{i=1}^{|\mathcal{V}|} \log p_\theta(X_i, E_i \mid X_{<i}, E_{<i}). \tag{2}$$

At each step i, we use all nodes that are generated before i, their attributes $X_{<i}$, and the structure (edges) between these nodes $E_{<i}$ to generate a new node i, including both its attribute X_i and its connections with existing nodes E_i.

Essentially, the objective in Eq. 2 describes the autoregressive generative process of an attributed graph. The question becomes: *how to model the conditional probability* $p_\theta(X_i, E_i | X_{<i}, E_{<i})$*?*


Phase I: Pre-process the input graph
Phase II: Pre-train by maximizing the likelihood of attributes and masked edges
1
2
3
4
5
+1
-1
X3
X4
X5
nodes with attributes
masked edges
attribute generation node
edge generation node
maximize likelihood

(a) Determine the node permutation order

(b) Determine masked edges and their directions

(c) Generate attributes and masked edges for node 3

(d) Generate attributes and masked edges for node 4

(e) Generate attributes and masked edges for node 5

Figure 2: An illustrative example of the proposed attributed graph generation procedure.

3.3 Factorizing Attributed Graph Generation

To compute $p_\theta(X_i, E_i | X_{<i}, E_{<i})$, one naive solution could be to simply assume that X_i and E_i are independent, that is,

$$p_\theta(X_i, E_i | X_{<i}, E_{<i}) = p_\theta(X_i | X_{<i}, E_{<i}) \cdot p_\theta(E_i | X_{<i}, E_{<i})$$

With such decomposition, for each node, the dependency between its attributes and connections are completely neglected. However, the ignored dependency is the core property of attributed graphs and also the foundation of convolutional aggregation in GNNs. Therefore, such a naive decomposition cannot provide informative guidance for pre-training GNNs.

To address this issue, we present the dependency-aware factorization mechanism for the attributed graph generation process. Specifically, when estimating a new node's attributes, we are given its structure information, and vice versa. During the process, a part of the edges has already been observed (or generated). Then the generation can be decomposed into two coupled parts:

- given the observed edges, generate node attributes;
- given the observed edges and generated node attributes, generate the remaining edges.

In this way, the model can capture the dependency between the attributes and structure for each node.

Formally, we define a variable o to denote the index vector of all the observed edges within E_i. Thus, $E_{i,o}$ denotes the observed edges. Similarly, $\neg o$ denotes the index of all the masked edges, which are to be generated. With this, we can rewrite the conditional probability as an expected likelihood over all observed edges:

$$\begin{aligned}
&p_\theta(X_i, E_i \mid X_{<i}, E_{<i}) \\
=& \sum\nolimits_o p_\theta(X_i, E_{i,\neg o} \mid E_{i,o}, X_{<i}, E_{<i}) \cdot p_\theta(E_{i,o} \mid X_{<i}, E_{<i}) \\
=& \mathbb{E}_o \Big[p_\theta(X_i, E_{i,\neg o} \mid E_{i,o}, X_{<i}, E_{<i}) \Big] \\
=& \mathbb{E}_o \Big[\underbrace{p_\theta(X_i \mid E_{i,o}, X_{<i}, E_{<i})}_{\text{1) generate attributes}} \cdot \underbrace{p_\theta(E_{i,\neg o} \mid E_{i,o}, X_{\le i}, E_{<i})}_{\text{2) generate edges}} \Big]. \qquad (3)
\end{aligned}$$

This factorization design is able to model the dependency between node i's attributes X_i and its associated connections E_i. The first term $p_\theta(X_i \mid E_{i,o}, X_{<i}, E_{<i})$ denotes the generation of attributes for node i. Based on the observed edges $E_{i,o}$, we gather the target node i's neighborhood information to generate its attributes X_i. The second term $p_\theta(E_{i,\neg o} \mid E_{i,o}, X_{\le i}, E_{<i})$ denotes the generation of masked edges. Based on both the observed edges $E_{i,o}$ and the generated attributes X_i, we generate the representation of the target node i and predict whether each edge within $E_{i,\neg o}$ exists.

A graph generation example. We intuitively show how the proposed factorization-based graph generation process works. Take, for example, an academic graph, if we would like to generate one paper node, whose title is considered as its attribute, while this paper node is connected to its authors, published venue, and cited papers. Based on some observed edges between this paper and some of its authors, our generation process first generates its title. Then, based on both the observed edges and generated title, we predict its remaining authors, published venue, and references. In this way, this process models the interaction between the paper's attribute (title) and structure (observed and remaining edges) to complete the generation task, bringing in informative signals for pre-training GNNs over the academic graph.

So far, we factorize the attributed graph generation process into a node attribute generation step and an edge generation step. The question we need to answer here is: *How to efficiently pre-train GNNs by optimizing both attribute and edge generation tasks?*

3.4 Efficient Attribute and Edge Generation

For the sake of efficiency, it is desired to compute the loss of attribute and edge generations by running the GNN only once for the input graph. In addition, we expect to conduct attribute generation and edge generation simultaneously. However, edge generation requires node attributes as input, which can be leaked to attribute generation. To avoid information leakage, we design to separate each node into two types:

- Attribute Generation Nodes. We mask out the attributes of these nodes by replacing their attributes with a dummy token and learn a shared vector X^{init} to represent it[†]. This is equivalent to the trick of using the [Mask] token in the masked language model [4].
- Edge Generation Nodes. For these nodes, we keep their attributes and put them as input to the GNN.

[†] X^{init} has the same dimension as X_i and can be learned during pre-training.

We then input the modified graph to the GNN model and generate the output representations. We use h^{Attr} and h^{Edge} to represent the output embeddings of Attribute Generation and Edge Generation Nodes, respectively. As the attributes of Attribute Generation Nodes are masked out, h^{Attr} in general contains less information than h^{Edge}. Therefore, when conduct the GNN message passing, we only use Edge Generation Nodes' output h^{Edge} as outward messages. The representations of the two sets of nodes are then used to generate attributes and edges with different decoders.

For Attribute Generation, we denote its decoder as $Dec^{Attr}(\cdot)$, which takes h^{Attr} as input and generates the masked attributes. The modeling choice depends on the type of attributes. For example, if the input attribute of a node is text, we can use the text generator model (e.g., LSTM) to generate it. If the input attribute is a standard vector, we can apply a multi-layer Perceptron to generate it. Also, we define a distance function as a metric between the generated attributes and the real ones, such as perplexity for text or L2-distance for vectors. Thus, we calculate the attribute generation loss via:

$$\mathcal{L}_i^{Attr} = Distance\big(Dec^{Attr}(h_i^{Attr}), X_i\big). \tag{4}$$

By minimizing the distance between the generated and masked attributes, it is equivalent to maximize the likelihood to observe each node attribute, i.e., $p_\theta(X_i \mid E_{i,o}, X_{<i}, E_{<i})$, and thus the pre-trained model can capture the semantic of this graph.

For Edge Generation, we assume that the generation of each edge is independent with others, so that we can factorize the likelihood:

$$p_\theta(E_{i,\neg o} \mid E_{i,o}, X_{\le i}, E_{<i}) = \prod_{j^+ \in E_{i,\neg o}} p_\theta(j^+ \mid E_{i,o}, X_{\le i}, E_{<i}). \tag{5}$$

Next, after getting the Edge Generation node representation h^{Edge}, we model the likelihood that node i is connected with node j by $Dec^{Edge}(h_i^{Edge}, h_j^{Edge})$, where Dec^{Edge} is a pairwise score function. Finally, we adopt the negative contrastive estimation to calculate the likelihood for each linked node j^+. We prepare all the unconnected nodes as S_i^- and calculate the contrastive loss via

$$\mathcal{L}_i^{Edge} = -\sum_{j^+ \in E_{i,\neg o}} \log \frac{\exp\big(Dec^{Edge}(h_i^{Edge},\ h_{j^+}^{Edge})\big)}{\sum_{j \in S_i^- \cup \{j^+\}} \exp\big(Dec^{Edge}(h_i^{Edge},\ h_j^{Edge})\big)} \tag{6}$$

By optimizing $\mathcal{L}^{Edge}$, it is equivalent to maximizing the likelihood of generating all the edges, and thus the pre-trained model is able to capture the intrinsic structure of the graph.

Figure 2 illustrates the attributed graph generation process. Specifically: (a) We determine the node permutation order π for the input graph. (b) We randomly select a portion of the target node's edges as observed edges $E_{i,o}$ and the remaining as masked edges $E_{i,\neg o}$ (grey dashed lines with cross). We delete masked edges in the graph. (c) We separate each node into the Attribute Generation and Edge Generation nodes to avoid information leakage. (d) After the pre-processing, we use the modified adjacency matrix to calculate the representations of node 3,4 and 5, including both their Attribute and Edge Generation Nodes. Finally, as illustrated in (d)–(e), we train the GNN model via the attribute prediction and masked edge prediction task for each node in parallel. The overall pipeline of GPT-GNN is illustrated in Algo. 1 (See Appendix B for details).

Algorithm 1 The GPT-GNN Pre-Training Framework

```
Require: Input Attributed Graph $G$, Graph Sampler $Sampler(\cdot)$.
Ensure:
Initialize the GNN model as $f_\theta$, the attribute generation decoder
  as $Dec^{Attr}$, and the edge generation decoder as $Dec^{Edge}$.
Initialize the adaptive node embedding queue $Q = \{\}$ and the
  attribute vector $h^{init}$.
for each sampled graph $\hat{G} \in Sampler(G)$ do
    For each node, sample the observed edge index $o$ and masked
      edges $\neg o$, and delete masked edges $E_{i,\neg o}$ accordingly.
    Separate each node into the Attribute Generation and Edge
      Generation nodes. Replace the input to Attribute Genera-
      tion node as $h_{init}$. Apply GNN $f_\theta$ to get two sets of node
      embeddings $h^{Attr}$ and $h^{Edge}$ for each node in the graph.
    for node $i$ with attributes $X_i$ and masked edges $E_{i,\neg o}$ do
        Calculate the attribute generation loss $\mathcal{L}^{Attr}$ by Eq. 4
        Prepare negative samples $S_i^-$ for edge generation by con-
          catenating unconnected nodes and adaptive queue $Q$.
        Calculate the edge generation loss $\mathcal{L}^{Edge}$ by Eq. 6
    end for
    Optimize $\theta$ by minimizing $\mathcal{L}^{Attr}$ and $\mathcal{L}^{Edge}$.
    Update $Q$ by adding in $h^{Edge}$ and popping out most outdated
      embeddings.
end for
return Pre-trained model parameters $\theta^*$ for downstream tasks
```

3.5 GPT-GNN for Heterogeneous & Large Graphs

In this section, we discuss how to apply GPT-GNN to pre-train for large-scale and heterogeneous graphs, which can be of practical use for modeling real-world complex systems [7, 30], such as academic graphs, product graphs, IoT networks, and knowledge graphs.

Heterogeneous graphs. Many real-world graphs are heterogeneous, meaning that they contain different types of nodes and edges. For heterogeneous graphs, the proposed GPT-GNN framework can be straightforwardly applied to pre-train heterogeneous GNNs. The only difference is that each type of nodes and edges may have its own decoder, which is specified by the heterogeneous GNNs rather than the pre-training framework. All the other components remain exactly the same.

Large-scale graphs. To pre-train GNNs on graphs that are too large to fit into the hardware, we sample subgraphs for training. In particular, we propose to sample a dense subgraph from homogeneous and heterogeneous graphs by using the LADIES algorithm [45] and its heterogeneous version HGSampling [15], respectively. Both methods theoretically guarantee that the sampled nodes are highly interconnected with each other and maximally preserve the structural information.

To estimate the contrastive loss in Eq. 6, it is required to go over all nodes of the input graph. However, we only have access to the sampled nodes in a subgraph for estimating this loss, making the (self-)supervision only focus on local signals. To alleviate this issue, we propose the *Adaptive Queue*, which stores node representations in previously-sampled subgraphs as negative samples. Each time we process a new subgraph, we progressively update this queue by adding the latest node representations and remove the oldest ones. As the model parameters will not be updated rigorously, the negative samples stored in the queue are consistent and accurate. The *Adaptive Queue* enables us to use much larger negative sample pools S_i^-. Moreover, the nodes across different sampled sub-graphs can bring in the global structural guidance for contrastive learning.

4 EVALUATION

To evaluate the performance of GPT-GNN, we conduct experiments on the Open Academic Graph (OAG) and Amazon Recommendation datasets. To evaluate the generalizability of GPT-GNN, we consider different transfer settings—time transfer and field transfer—which are of practical importance.

4.1 Experimental Setup

Datasets and Tasks. We conduct experiments on both heterogeneous and homogeneous graphs. For heterogeneous graphs, we use the Open Academic Graph and Amazon Review Recommendation data. For homogeneous graphs, we use the Reddit dataset [12] and the paper citation network extracted from OAG. All datasets are publicly available and the details can be found in Appendix A.

Open Academic Graph (OAG) [34, 38, 44] contains more than 178 million nodes and 2.236 billion edges. It is the largest publicly available heterogeneous academic dataset to date. Each paper is labeled with a set of research topics/fields (e.g., Physics and Medicine) and the publication date ranges from 1900 to 2019. We consider the prediction of Paper–Field, Paper–Venue, and Author Name Disambiguation (Author ND) as three downstream tasks [7, 15]. The performance is evaluated by MRR—a widely adopted ranking metric [19].

Amazon Review Recommendation Dataset (Amazon) [23] contains 82.8 million reviews, 20.9 million users, and 9.3 million products. The reviews are published from 1996 to 2018. Each review consists of a discrete rating score from 1 to 5 and a specific field, including book, fashion, etc. For downstream tasks, we predict the rating score as a five-class classification task within the Fashion, Beauty, and Luxury fields. We use micro F1-score as the evaluation metric.

The base GNN model. On the OAG and Amazon datasets, we use the state-of-the-art heterogeneous GNN—Heterogeneous Graph Transformer (HGT) [15]—as the base model for GPT-GNN. Furthermore, we also use other (heterogeneous) GNNs as the base model to test our generative pre-training framework.

Implementation details. For all base models, we set the hidden dimension as 400, the head number as 8, and the number of GNN layers as 3. All of them are implemented using the PyTorch Geometric (PyG) package [8].

We optimize the model via the AdamW optimizer [21] with the Cosine Annealing Learning Rate Scheduler [20] with 500 epochs and select the one with the lowest validation loss as the pre-trained model. We set the adaptive queue size to be 256.

During downstream evaluation, we fine-tune the model using the same optimization setting for 200 epochs as that in pre-training. We train the model on the downstream tasks for five times and report the mean and standard deviation of test performance.

Pre-training baselines. There exist several works that propose unsupervised objectives over graphs, which can potentially be used to pre-train GNNs. We thus compare the proposed GPT-GNN framework with these baselines:

- *GAE* [16], which denotes graph auto-encoders, focuses on a traditional link prediction task. It randomly masks out a fixed proportion of the edges and asks the model to reconstruct these masked edges.
- *GraphSAGE (unsp.)* [12] forces connected nodes to have similar output node embeddings. Its main difference with GAE lies in that it does not mask out the edges during pre-training.
- *Graph Infomax* [37] tries to maximize the local node embeddings with global graph summary embeddings. Following its setting for a large-scale graph, for each sampled subgraph, we shuffle the graph to construct negative samples.

In addition, we also evaluate the two pre-training tasks in GPT-GNN by using each one of them alone, that is, attribute generation—*GPT-GNN (Attr)*—and edge generation—*GPT-GNN (Edge)*.

4.2 Pre-Training and Fine-Tuning Setup

The goal of pre-training is to transfer knowledge learned from numerous unlabeled nodes of a large graph to facilitate the downstream tasks with a few labels. Specifically, we first pre-train a GNN and use the pre-trained model weights to initialize models for downstream tasks. We then fine-tune the models with the downstream task specific decoder on the training (fine-tuning) set and evaluate the performance on the test set.

Broadly, there are two different setups. The first one is to pre-train and fine-tune on exactly the same graph. The second one is relatively more practical, which is to pre-train on one graph and fine-tune on unseen graphs of the same type as the pre-training one. Specifically, we consider the following three graph transfer settings between the pre-training and fine-tuning stages:

- *Time Transfer*, where we use data from different time spans for pre-training and fine-tuning. For both OAG and Amazon, we use data before 2014 for pre-training and data since 2014 for fine-tuning.
- *Field Transfer*, where we use data from different fields for pre-training and evaluating. In OAG, we use papers in the field of computer science (CS) for downstream fine-tuning and use all papers in the remaining fields (e.g., Medicine) for pre-training. In Amazon, we pre-train on products in Arts, Crafts, and Sewing, and fine-tune on products in Fashion, Beauty, and Luxury.
- *Time + Field Transfer*, where we use the graph of particular fields before 2014 to pre-train the model and use the data from other fields since 2014 for fine-tuning. Intuitively, this combined transfer setting is more challenging than the transfer of time or field alone. For example, we pre-train on the Medicine graph before 2014 and fine-tune on the CS graph since 2014.

During fine-tuning, for both datasets, we choose nodes from 2014 to 2016 for training, 2017 for validation, and since 2018 for testing. To meet the assumption that training data is usually scarce, we only provide 10% of the labels for training (fine-tuning) by default, while the ablation study over different data percentages is also conducted.

	Downstream Dataset	OAG			Amazon		
	Evaluation Task	Paper–Field	Paper–Venue	Author ND	Fashion	Beauty	Luxury
	No Pre-train	.346±.149	.598±.122	.813±.105	.586±.074	.546±.071	.494±.067
Field Transfer	GAE	.403±.114	.626±.093	.836±.084	.610±.070	.568±.066	.516±.071
	GraphSAGE (unsp.)	.368±.125	.609±.096	.818±.092	.597±.065	.554±.061	.509±.052
	Graph Infomax	.387±.112	.612±.097	.827±.084	.604±.063	.561±.063	.506±.074
	GPT-GNN (Attr)	.396±.118	.623±.105	.834±.086	.621±.053	.576±.056	.528±.061
	GPT-GNN (Edge)	.413±.109	.635±.096	.842±.093	.616±.060	.570±.059	.520±.047
	GPT-GNN	**.420±.107**	**.641±.098**	**.848±.102**	**.625±.055**	**.577±.054**	**.531±.043**
Time Transfer	GAE	.384±.117	.619±.101	.828±.095	.603±.065	.562±.063	.510±.071
	GraphSAGE (unsp.)	.352±.121	.601±.105	.815±.093	.594±.067	.553±.069	.501±.064
	Graph Infomax	.369±.116	.606±.102	.821±.089	.599±.063	.558±.060	.503±.063
	GPT-GNN (Attr)	.374±.114	.614±.098	.826±.089	.614±.057	**.573±.053**	.522±.051
	GPT-GNN (Edge)	.397±.105	.629±.102	.836±.088	.608±.055	.567±.038	.513±.058
	GPT-GNN	**.405±.108**	**.635±.101**	**.840±.093**	**.617±.059**	.572±.059	**.525±.057**
Time + Field Transfer	GAE	.371±.124	.611±.108	.821±.102	.596±.065	.554±.063	.505±.061
	GraphSAGE (unsp.)	.349±.130	.602±.118	.812±.097	.589±.071	.545±.068	.498±.064
	Graph Infomax	.360±.121	.600±.102	.815±.093	.591±.068	.550±.058	.501±.063
	GPT-GNN (Attr)	.364±.115	.609±.103	.824±.094	.608±.062	.569±.057	.517±.057
	— (w/o node separation)	.347±.128	.601±.102	.813±.108	.585±.068	.546±.062	.497±.062
	GPT-GNN (Edge)	.390±.116	.622±.104	.830±.105	.604±.058	.565±.057	.514±.047
	— (w/o adaptive queue)	.376±.121	.617±.115	.828±.104	.599±.068	.562±.065	.509±.062
	GPT-GNN	**.397±.112**	**.628±.108**	**.833±.102**	**.610±.054**	**.572±.063**	**.521±.049**

Table 1: Performance of different downstream tasks on OAG and Amazon by using different pre-training frameworks with the heterogeneous graph transformer (HGT) [15] as the base model. 10% of labeled data is used for fine-tuning.

During pre-training, we randomly select a subset of the data as the validation set.

4.3 Experimental Results

We summarize the performance of downstream tasks with different pre-training methods on OAG and Amazon in Table 1. As discussed above, we setup three different transfer settings between pre-training and fine-tuning stages: Field Transfer, Time Transfer, and Field + Time Combined Transfer, as organized in three different blocks in the Table.

Overall, the proposed GPT-GNN framework significantly enhances the performance for all downstream tasks on both datasets. On average, GPT-GNN achieves relative performance gains of 9.1% and 5.7% over the base model without pre-training on OAG and Amazon, respectively. Moreover, it consistently outperforms other pre-training frameworks, such as Graph Infomax, across different downstream tasks for all three transfer settings on both datasets.

Different transfer settings. Observed from Table 1, the performance gain lifted by pre-training under the field transfer is higher than that under the time transfer, and the time + field combined transfer is the most challenging setting as evident in the least performance gain brought by pre-training. Nonetheless, under the combined transfer, GPT-GNN still achieves 7.4% and 4.6% performance gains on both datasets, respectively. Altogether, the results suggest that *the proposed generative pre-training strategy enables the GNN model to capture the generic structural and semantic knowledge of the input graph, which can be used to fine-tune on the unseen part of the graph data.*

Ablation studies on pre-training tasks. We analyze the effectiveness of the two pre-training tasks in GPT-GNN—attribute generation and edge generation—by examining which of them is more beneficial for the pre-training framework and, by extension, downstream tasks. In Table 1, we report the performance of GPT-GNN by using attribute generation and edge generation alone, that is, GPT-GNN (Attr) and GPT-GNN (Edge). On OAG, the average performance gains by GPT-GNN (Attr) and GPT-GNN (Edge) are 4.7% and 7.9%, respectively, suggesting that Edge Generation is a more informative pre-training task than Attribute Generation in GPT-GNN. However, we have an opposite observation for Amazon, on which the performance improved by Attribute Generation is 5.2% in contrast to the 4.1% improvement lifted by Edge Generation. *This suggests that the GPT-GNN framework benefits differently from attribute and edge generations on different datasets. However, combining the two pre-training tasks together produces the best performance on both cases.*

We further compare the Edge Generation task against other edge-based pre-training methods—GAE and GraphSage—in Table 1. On OAG, the performance improvements brought by GPT-GNN's edge generation, GAE, and GraphSage over no pre-training are 7.9%, 5.6%, and 1.4%, respectively. On Amazon, the gains are 5.2%, 3.1%,

Model	HGT	GCN	GAT	RGCN	HAN
No Pre-train	**.346**	.327	.318	.296	.332
GPT-GNN	**.420**	.359	.382	.351	.406
Relative Gain	21.4%	9.8%	20.1%	18.9%	22.3%

Table 2: Compare the pre-training Gain with different GNN architectures. Evaluate on OAG, Paper-Field Task, under Combined Transfer setting with 10% training data.

and 1.3%, respectively. From the comparisons, we have the following observations. First, both GAE and GPT-GNN's edge generation offer better results than GraphSage on both datasets, demonstrating that masking on edges is an effective strategy for self-supervised graph representation learning. Without edge masking, the model simply retains a similar embedding for connected nodes, as the label we would like to predict (whether two nodes are linked) has already been encoded in the input graph structure. Such information leakage will downgrade the edge prediction task to a trivial problem. Second, the proposed Edge Generation task consistently outperforms GAE. The main advantage of GPT-GNN's edge generation comes from that it learns to generate missing edges autoregressively and thus can capture the dependencies between the masked edges, which are discarded by GAE. *In summary, the results suggest that the proposed graph generation tasks can give informative self-supervision for GNN pre-training.*

Ablation studies on the base GNN. We investigate whether the other GNN architectures can benefit from the proposed pre-training framework. Therefore, in addition to HGT [15], we try GCN [17], GAT [36], RGCN [28], and HAN [39] as the base model. Specifically, we pre-train them on OAG and then use the paper-field prediction task under the combined transfer setting with 10% of training data for fine-tuning and testing. Model-independent hyper-parameters, such as the hidden dimension size and optimization, are set the same. The results are reported in in Table 2. We can observe that 1) HGT achieves the best performance among all non pre-trained GNN models; 2) GPT-GNN with HGT generates the most promising results for the concerned downstream task; and 3) *more importantly, the proposed GPT-GNN pre-training framework can enhance the downstream performance for all the GNN architectures.*

Ablation studies on the node separation and adaptive queue. Finally, we examine the effectiveness of the two design choices of GPT-GNN, i.e., node separation and adaptive queue. The node separation is designed for alleviating the information leakage problem for the Attribute Generation task. Without this component, the attributes would appear in the input and thus the pre-training method would only need to maintain the input features for output. In other words, it cannot learn any knowledge of the input graph that could be transferred to downstream tasks and thus affect the results negatively. From Table 1, we can see that the attribute generation based pre-training model suffers from the removal of the node separation component (w/o node separation), and in many cases, its performance is even worse than the ones without pre-training. *This demonstrates the significance of this node separation design in avoiding attribute information leakage.*

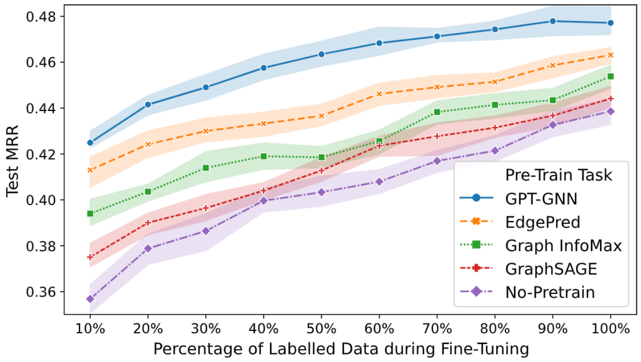


Figure 3: Compare pre-training tasks with different training data size. Evaluated by the paper–field prediction task on OAG under the field transfer setting.

Downstream Dataset	OAG (citation)	Reddit
No Pre-train	.281±.087	.873±.036
GAE	.296±.095	.885±.039
GraphSAGE (unsp.)	.287±.093	.880±.042
Graph Infomax	.291±.086	.877±.034
GPT-GNN	**.309±.081**	**.896±.028**

Table 3: Downstream performance on homogeneous graphs, including the paper citation network in OAG and Reddit.

The adaptive queue is designed for alleviating the gap between the sampled graphs and the full graph. Similarly, we conduct the ablation study by removing it from the Edge Generation based pre-training model and from Table 1, we witness the consistent performance drops for all tasks—GPT-GNN (Edge) vs. (w/o adaptive queue). *This indicates that adding more negative samples by using the adaptive queue is indeed helpful to the pre-training framework.*

Training data size. In Figure 3, we examine whether the proposed GPT-GNN method can generalize well with different training data size during fine-tuning, i.e., from 10% to 100%. First, we can observe that GPT-GNN and other pre-training frameworks consistently improve the downstream task performance with more labeled training data. Second, it is clear that GPT-GNN performs the best among all pre-training tasks/frameworks. Finally, we can see that *with the pre-trained model, fine-tuning with only 10–20% of data (the two leftmost blue circles) generates comparative performance to the supervised learning with all 100% of training data (the rightmost purple diamond), demonstrating the superiority of GNN pre-training, especially when the label is scarce.*

Results for homogeneous graphs. In addition to heterogeneous graphs, we also test whether the GPT-GNN pre-training framework can benefit downstream tasks on homogeneous graphs. Specifically, we pre-train and fine-tune on two homogeneous graphs: 1) the paper citation network extracted from the field of CS in OAG, on which the topic of each paper is predicted; 2) the Reddit network consisting of Reddit posts, on which the community of each post is inferred. As there is only one type of nodes and edges in homogeneous graphs, we require one single set of edge and attribute decoders for pre-training. HGT is used as the base pre-training model by ignoring its heterogeneous components. The fine-tuned results with 10% labeled data are presented in Table 3. We can observe that the downstream tasks on both homogeneous graphs can benefit from all pre-training frameworks, among which the proposed GPT-GNN offers the largest performance gains.

5 CONCLUSION

In this work, we study the problem of graph neural network pre-training. We present GPT-GNN—a generative GNN pre-training framework. In GPT-GNN, we design the graph generation factorization to guide the base GNN model to autoregressively reconstruct both the attributes and structure of the input graph. Furthermore, we propose to separate the attribute and edge generation nodes to avoid information leakage. In addition, we introduce the adaptive node representation queue to mitigate the gap between the likelihoods over the sampled graph and the full graph. The pre-trained GNNs with fine-tuning over few labeled data can achieve significant performance gains on various downstream tasks across different datasets. Also, GPT-GNN is robust to different transfer settings between pre-training and fine-tuning. Finally, we find that fine-tuning the generative pre-trained GNN model with 10–20% of labeled data offers comparative performance for downstream tasks to the supervised GNN model with 100% of training data.

Acknowledgements. This work is partially supported by NSF III-1705169, NSF CAREER Award 1741634, NSF 1937599, DARPA HR00112090027, DARPA N660011924032, Okawa Foundation Grant, and Amazon Research Award.

REFERENCES

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203* (2013).

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. *arxiv:2002.05709* (2020).

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. (2009).

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL 2019*.

[5] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML 2014*.

[6] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD 2017*.

[7] Yuxiao Dong, Ziniu Hu, Kuansan Wang, Yizhou Sun, and Jie Tang. 2020. Heterogeneous Network Representation Learning. In *IJCAI*.

[8] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *ICLR Workshop* (2019).

[9] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML 2017*.

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR 2014*.

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD 2016*.

[12] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS 2017*.

[13] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2019. Momentum contrast for unsupervised visual representation learning. *arXiv:1911.05722* (2019).

[14] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. 2020. Strategies for Pre-training Graph Neural Networks. In *ICLR 2020*.

[15] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *WWW 2020*.

[16] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *arXiv:1611.07308* (2016).

[17] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR 2017*.

[18] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. 2019. Efficient Graph Generation with Graph Recurrent Attention Networks. In *NeurIPS 2019*.

[19] Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval*. Springer.

[20] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR 2017*.

[21] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *ICLR 2019*.

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS 2013*.

[23] Jianmo Ni, Jiacheng Li, and Julian J. McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *EMNLP 2019*.

[24] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. 2016. Context Encoders: Feature Learning by Inpainting. In *CVPR 2016*.

[25] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP 2014*.

[26] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM '18*. 459–467.

[27] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[28] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *ESWC 2018*.

[29] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2020. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *ICLR 2020*.

[30] Yizhou Sun and Jiawei Han. 2012. *Mining Heterogeneous Information Networks: Principles and Methodologies*. Morgan & Claypool Publishers.

[31] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. In *VLDB 2011*.

[32] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S. Yu, and Xiao Yu. 2012. Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. In *KDD 2012*.

[33] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW 2015*.

[34] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD 2008*.

[35] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *arXiv:1807.03748* (2018).

[36] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR 2018*.

[37] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR 2019*.

[38] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft Academic Graph: When experts are not enough. *Quantitative Science Studies* 1, 1 (2020), 396–413.

[39] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. 2019. Heterogeneous Graph Attention Network. In *WWW 2019*.

[40] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Transformers: State-of-the-art Natural Language Processing. arXiv:cs.CL/1910.03771

[41] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *NeurIPS 2019*.

[42] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD 2018*.

[43] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *ICML 2018*.

[44] Fanjin Zhang, Xiao Liu, Jie Tang, Yuxiao Dong, Peiran Yao, Jie Zhang, Xiaotao Gu, Yan Wang, Bin Shao, Rui Li, and Kuansan Wang. 2019. OAG: Toward Linking Large-scale Heterogeneous Entity Graphs. In *KDD 2019*.

[45] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. In *NeurIPS 2019*.

A DATASET DETAILS

We mainly use *Open Academic Graph (OAG)* and *Amazon Review Recommendation Dataset (Amazon)* for evaluation. Both are widely used heterogeneous graph [30–32]. Here we introduce their statistics, schema and how we prepare the attributes and tasks in detail.

Open Academic Graph (OAG) [34, 38, 44] consists of five types of nodes: 'Paper', 'Author', 'Field', 'Venue', and 'Institute', and 14 types of edges between these nodes. The schema and meta relations are illustrated in Figure 4(a). For example, the 'Field' nodes in the OAG are categorized into six levels from L_0 to L_5, which are organized with a hierarchical tree (We use 'is_organized_in' to represent this hierarchy). Therefore, we differentiate the 'Paper–Field' edges in the corresponding field levels. Besides, we differentiate the different author orders (i.e., the first author, the last one, and others) and venue types (i.e., journal, conference, and preprint) as well. Finally, the 'Self' type corresponds to the self-loop connection, which is widely added in GNN architectures. Despite 'Self' and 'CoAuthor' edge relationships, which are symmetric, all other edge types X have a reverse edge type X^{-1}.

For downstream tasks, we choose the following three tasks: the prediction of Paper–Field (L2), Paper–Venue, and Author Disambiguation. In the first two tasks, we give a model a paper and want it to predict the correct fields it belongs to or the venue it is published at. We model these three tasks as node classification problem, where we use GNNs to get the contextual node representation of the paper and use a softmax output layer to get its classification results. For author disambiguation, we pick all the authors with the same name, and the papers that link to one of these same-name authors. The task is to conduct link prediction between papers and candidate authors. After getting the paper and author node representations from GNNs, we use a Neural Tensor Network to get the probability of each author-paper pair to be linked.

For input attributes of heterogeneous graph, as we don't assume the attribute of each data type belongs to the same distribution, and we are free to use the most appropriate attributes to represent each type of node. For paper and author nodes, the node numbers are extremely large. Therefore, traditional node embedding algorithms are not suitable for extracting attributes for them. We, therefore, resort to the paper titles for extracting attributes. For each paper, we get its title text and use a pre-trained XLNet [40, 41] to get the representation of each word in the title. We then average them weighted by each word's attention to get the title representation for each paper. The initial attribute of each author is simply an average of his/her published papers' embeddings. For field, venue and institute nodes, the node numbers are small and we use the metapath2vec model [6] to train their node embeddings by reflecting the heterogeneous network structures.

Amazon Review Recommendation Dataset (Amazon) [23] consists of three types of nodes, including reviews (ratings and text), users and products, and some other meta-types of the product, including color, size, style and quantity. The schema and meta relations are illustrated in Figure 4(b). Compared to a general user-item bipartite graph, this dataset have review in between, each is associated with text information and a rating from 1 to 5. The reviews are also associated with those product meta-type descriptions. For simplicity, we only consider these review-type link as 'categorize_in' type. Thus, there are totally three types of relations in this graph.

For downstream task, we choose the rating classification for each new review. Since the problem is a node-level multi-class classification, we use GNNs to get the contextual node representation of the review and use a softmax output layer to get 5-class prediction.

For input attributes of Amazon, we also use a pre-trained XLNet to get each review embedding, and the attributes for all the other nodes are simple an average of its associated review's embeddings.

B OVERALL PIPELINE OF GPT-GNN

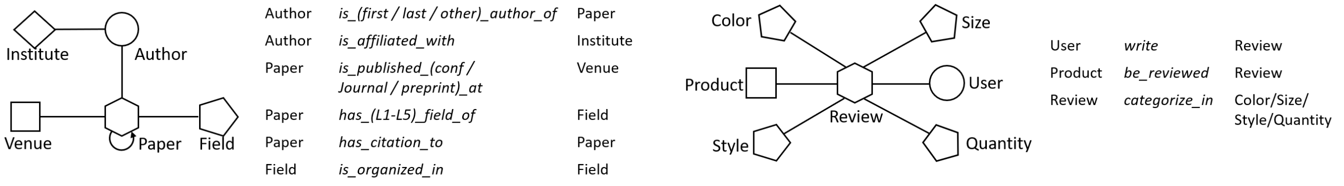
The overall pipeline of GPT-GNN is illustrated by Algorithm 1. Given an attributed graph G, we each time sample a subgraph $\hat{G}$ as training instance for generative pre-training. The first step is to determine the node permutation order π. To support parallel training, we want to conduct the forward pass for a single run and get the representation of the whole graph, so that we can simultaneously calculate the loss for each node, instead of processing each node recursively. Therefore, we remove all the edges from nodes with higher order to those with lower order according to π, which means each node can only receive information from nodes with lower order. In this way, they won't leak the information to the autoregressive generative objective, and thus we only need one single round to get node embeddings of the whole graph, which can directly be utilized to conduct generative pre-training.

Afterwards, we need to determine the edges to be masked out. For each node, we get all its outward edges, randomly select a set of edges to be masked out. This corresponds to line 4. Next, we conduct node separation and get contextualized node embeddings for the whole graph in line 5, which will be utilized to calculate generative loss. line 7-9. For both OAG and Amazon, the main nodes that contain meaningful attributes are paper and review nodes, which both have text feature as input. Thus, we only consider them for Attribute Generation, with a 2-layer LSTM as decoder. Note that in line 8, we prepare the negative samples by aggregating both the unconnected nodes within this sampled graph and the previously calculated embeddings stored in the adaptive queue Q. This can mitigate the gap between optimizing over sampled graph with over the whole graph. Finally, we optimize the model and update the adaptaive queue in line 11-12. Afterwards, we can use the pre-trained model as initialization, to fine-tune on other downstream tasks.

C IMPLEMENTATION DETAILS AND CONVERGENCE CURVES

We use a Tesla K80 to run both pre-training and downstream tasks. For graph sampling, we follow the HGSampling [15, 45] to sample subgraph over large-scale heterogeneous graph. For each type of node, we sample 128 nodes per layer. We repeat sampling for 6 times for OAG and average sampled nodes in the sub-graph is 3561 nodes. We repeat for 8 times for Amazon, and average sampled nodes is 1478. For each batch, we sample 32 graphs to conduct generative pre-training. During GPU training, we conduct multi-process sampling to prepare the pre-training data. Such CPU-GPU cooperation can help us save the sampling time.

We here illustrate the convergence curves for pre-training and fine-tuning. For pre-training, as illustrated in Figure 6, we show the



Author	*is_(first / last / other)_author_of*	Paper
Author	*is_affiliated_with*	Institute
Paper	*is_published_(conf / Journal / preprint)_at*	Venue
Paper	*has_(L1-L5)_field_of*	Field
Paper	*has_citation_to*	Paper
Field	*is_organized_in*	Field

(a) Schema and meta relations of Open Academic Graph (OAG)

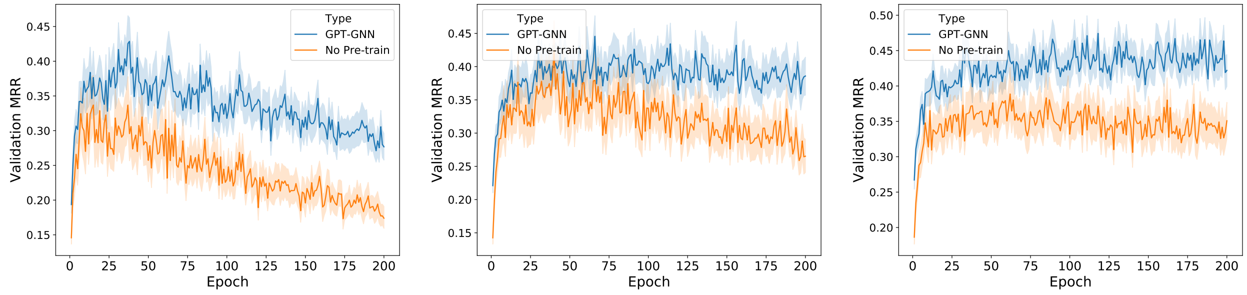
User	*write*	Review
Product	*be_reviewed*	Review
Review	*categorize_in*	Color/Size/ Style/Quantity

(b) Schema and meta relations of Amazon Review Recommendation Dataset (Amazon)

Figure 4: The schema and meta relations of Open Academic Graph and Amazon Review Recommendation Dataset.

Predict Paper Title	Groundtruth Paper Title
person recognition system using automatic probabilistic classification	person re-identification by probabilistic relative distance comparison
a novel framework using spectrum sensing in wireless systems	a secure collaborative spectrum sensing strategy in cyber physical systems
a efficient evaluation of a distributed data storage service storage	an empirical analysis of a large scale mobile cloud storage service
parameter control in wireless sensor networks networks networks	optimal parameter estimation under controlled communication over sensor networks
a experimental system for for to the analysis of graphics	an interactive computer graphics approach to surface representation

Table 4: Generated paper title samples. The left column is generated by GPT-GNN, and the right column is the groundtruth.



(a) Data Percentage: 10%

(b) Data Percentage: 20%

(c) Data Percentage: 50%

Figure 5: Fine-tuning convergence comparison of GPT-GNN with no-pretrain, under different training data percentage.

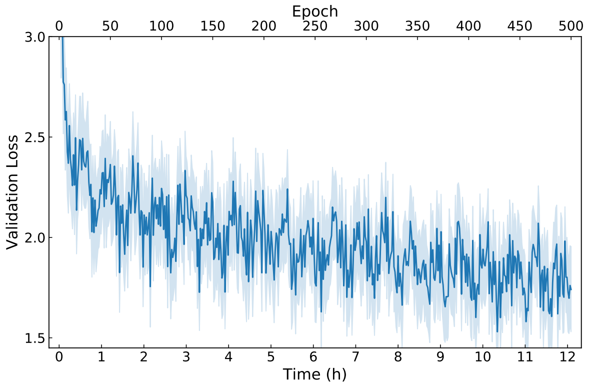


Figure 6: The pre-training convergence curve of GPT-GNN on OAG. It took about 10 hours (400 epochs) to converge.

pre-training validation error curve with respect to the epoch and time. Results show that the model's validation loss keep dropping instead of just finding a trivial solution very fast. This to some extent shows that the generative pre-training task is hard enough and can thus can guide the model to really capture the intrinsic structure of the graph data. It took about 12 hours for GPT-GNN to converge. For downstream tasks, we show the convergence curve utilizing our GPT-GNN with no-pretrain, with different data percentage. As is illustrated in Figure 5, GPT-GNN can always get a more generalized model than no-pretrain, and is more robust to over-fitting since a good initialization from pre-training.

D PAPER TITLE GENERATION EXAMPLES

For OAG, since our attribute generation task is oriented on the paper title, we'd like to see how well our GPT-GNN can learn to generate the title. The results are shown in table 4. We can see that the model can capture the main meaning of each paper to be predicted, only by looking at partial neighborhoods (note that we use Attribute Generation Node for this task, which replace the input attribute as a share vector). For example, for the first sentence, our model successfully predict the key words for this paper, including 'person recognition', 'probabilistic', etc. This shows that the graph itself contains rich semantic information, and explains why a pre-trained model can generalize well to downstream tasks.