석 사 학 위 논 문
Master's Thesis

그래프 표현의 정확한 학습을 위한 연구

Toward Accurate Learning of Graph Representations

2022

백 진 헌 (白 珍 憲 Baek, Jinheon)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

그래프 표현의 정확한 학습을 위한 연구

2022

백 진 헌

한 국 과 학 기 술 원

김재철AI대학원

그래프 표현의 정확한 학습을 위한 연구

백 진 헌

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2021년 12월 16일

심사위원장 황 성 주 (인)
심 사 위 원 신 기 정 (인)
심 사 위 원 양 은 호 (인)

Toward Accurate Learning of Graph Representations

Jinheon Baek

Advisor: Sung Ju Hwang

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in AI

Daejeon, Korea
December 16, 2021

Approved by

황 성주

Sung Ju Hwang
Professor in the Kim Jaechul Graduate School of AI

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MAI

백진헌. 그래프 표현의 정확한 학습을 위한 연구. 김재철AI대학원 . 2022년. 56+iv 쪽. 지도교수: 황성주. (영문 논문)
Jinheon Baek. Toward Accurate Learning of Graph Representations. Kim Jaechul Graduate School of AI . 2022. 56+iv pages. Advisor: Sung Ju Hwang. (Text in English)

초 록

그래프 데이터를 위한 학습 방법이 활발히 연구되고 있지만, 이전에 관측하지 못한 개체(노드)를 다루거나, 전체 그래프를 하나의 표현으로 나타내는 것은 도전적인 과제로 남아있다. 본 논문에서는 그래프의 정확한 표현을 가로막는 위 두 문제의 해결 방법을 소개한다. 첫 번째로 새롭게 관측한 개체의 기존 그래프 통합이라는 실제 문제를 정의하고, 문제 해결을 위한 트랙스덕티브 메타 학습을 제안한다. 또한 제안한 방법론이 지식 그래프와 약물 상호작용 그래프에서 새로운 지식과 약물을 잘 표현하여, 그래프 내외 개체들과 올바른 관계(엣지)를 형성하는 것을 보인다. 두 번째로 서로 다른 두 그래프를 다른 공간에 표현하기 위해 그래프 중복집합 풀링 방법론을 제안한 다음, 방법론이 이론적으로 단사 함수를 근사할 수 있음을 보인다. 더 나아가 그래프 분류, 복원 및 생성 문제에서 기존 방법론 대비 높은 성능을 달성하는 것을 보인다. 제안한 두 방법이 보다 정확한 그래프 표현에 크게 기여함을 기대한다.

핵 심 낱 말 기계 학습, 그래프 뉴럴 네트워크, 그래프 표현 학습, 메타 학습, 그래프 풀링, 관계 예측, 그래프 분류, 그래프 복원, 그래프 생성

Abstract

While machine learning algorithms and models for graph-structured data have been actively studied, the problems of handling new entities (nodes) and representing entire graphs remain challenging. My thesis focuses on how to tackle such problematic issues on a graph. In other words, we develop methods for representing unseen entities and entire graphs more accurately, summarized into two folds:

For the problem of representing unseen entities, we first introduce a realistic task of out-of-graph link prediction that aims to predict missing links for unseen entities. Then, to tackle this, we propose a transductive meta-learning framework that makes it possible to simulate the unseen during training. We validate our method on benchmark datasets for knowledge graph completion and drug-drug interaction prediction. The experimental results show that our method significantly outperforms existing baselines on the out-of-graph link prediction task, due to its effectiveness in accurately representing unseen entities.

For the problem of representing entire graphs, we aim to embed different graphs into distinct vectors. To do so, we consider the graph encoding problem as a multiset encoding problem, which allows for possibly repeating elements, since a graph may have redundant nodes. Then, over the multiset encoding scheme, we propose a graph multiset transformer that captures interaction among nodes, while reducing the size of the given graph, to obtain a compact yet entire graph representation. We theoretically prove that our method is as powerful as the Weisfeiler-Lehman graph isomorphism test, but also empirically show that it outperforms baselines on graph classification, reconstruction, and generation tasks.

We believe both of our approaches contribute to the optimal goal of accurate learning of real-world graphs, often evolving with unseen nodes and having a large number of nodes to capture at once.

Keywords Machine Learning, Graph Neural Networks, Graph Representation Learning, Meta Learning, Graph Pooling, Link Prediction, Graph Classification, Graph Reconstruction, Graph Generation

Contents

List of Tables

List of Figures

Chapter 1. Introduction

A graph can express a rich interaction between elements, which is widely used to denote a large number of interconnected systems, including but not limited to social science [1], physical system [2], and knowledge base [3], with a set of objects and their relations. To deal with such graph-structured data under an end-to-end learning scheme, graph neural networks (GNNs) have been proposed [4, 5], which generally represent the nodes by aggregating the features of their neighborhoods. In spite of their simplicity in representing nodes of the given graph, they have been broadly utilized for many graph-related tasks, such as node classification [6], link prediction [7], travel-time prediction [8], recommendation [9], to name a few.

However, despite such successes of graph neural networks, there are important remaining challenges that hurt the accurateness of graph representation learning. For example, in a real-world scenario, graphs often have an evolving nature, where new entities are consistently coming in with few links. Also, for tasks requiring a representation of an entire graph or a subgraph (e.g., graph classification [10] and graph retrieval [11]), we have to summarize the representation of multiple nodes. However, most of the existing node-level GNNs are insufficient to obtain accurate representations of emerging nodes and entire graphs.

Thus, in this thesis, we first focus on the problems of existing node-level GNNs for each task of the unseen node representation and the entire graph representation. After that, to tackle those limitations, we introduce two methods that can be easily coupled with existing node-level GNNs, yet make significant performance gains from them.

Specifically, in Chapter 2, we focus on the challenges of evolving graphs, such as knowledge graph and drug-drug interaction graph, where new entities emerge over time with few associated links to embed them. Then, to formally define this observed challenge, we introduce a novel problem of few-shot out-of-graph link prediction. After that, we solve the proposed problem with a transductive meta-learning framework, where the model learns to represent unseen entities that are simulated from seen entities during training, unlike the conventional learning scheme that cannot observe unseen at training. As an architecture choice for the meta-learning framework, we use an existing message-passing scheme of GNNs, thus showing that existing GNNs can accurately represent an unseen entity by extrapolating the knowledge from its associated seen entities, only with a change of GNNs' learning scheme.

On the other hand, in Chapter 3, we aim at obtaining a compact representation of an entire graph, for which effectively summarizing a set of node representations obtained from node-level GNNs is an important challenge. Notably, when representing an entire graph, one might further want to distinguish two different graphs in the representation space (i.e., obtaining different representations from different graphs), thus we further aim at obtaining the graph representation that is as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test [12]. In order to satisfy those two requirements (i.e., summarizing the whole node representations while distinguishing different graphs), we propose a transformer-based architecture that captures interaction among nodes, and then prove that it is at most as powerful as the WL test. Our model is easily coupled with existing node-level GNNs, but significantly improves the performance of existing GNNs on graph-level tasks (e.g., graph classification, reconstruction, and generation) with high memory and time efficiency.

Finally, in Chapter 4, we summarize the contributions and the potential directions of our work: toward accurate learning of graph representations on all scales from nodes, to edges, to graphs.

This Chapter is based on the work that is published at NeurIPS 2020 [13].

Chapter 2. Accurate Learning of Unseen Node Representations

How do we handle emerging entities, which are common in evolving graphs such as knowledge graphs, that are not seen during training? We provide the summary of the problem that we tackle, the method that we propose, and the results that we obtain, in below.

Many practical graph problems, such as knowledge graph construction and drug-drug interaction prediction, require to handle multi-relational graphs. However, handling real-world multi-relational graphs with Graph Neural Networks (GNNs) is often challenging due to their evolving nature, as new entities (nodes) can emerge over time. Moreover, newly emerged entities often have few links, which makes the learning even more difficult. Motivated by this challenge, we introduce a realistic problem of *few-shot out-of-graph link prediction*, where we not only predict the links between the seen and unseen nodes as in a conventional out-of-knowledge link prediction task but also between the unseen nodes, with only few edges per node. We tackle this problem with a novel transductive meta-learning framework which we refer to as *Graph Extrapolation Networks (GEN)*. GEN meta-learns both the node embedding network for inductive inference (seen-to-unseen) and the link prediction network for transductive inference (unseen-to-unseen). For transductive link prediction, we further propose a stochastic embedding layer to model uncertainty in the link prediction between unseen entities. We validate our model on multiple benchmark datasets for knowledge graph completion and drug-drug interaction prediction. The results show that our model significantly outperforms relevant baselines for out-of-graph link prediction tasks.

2.1 Introduction

Graphs have a strong expressive power to represent structured data, as they can model data into a set of nodes (objects) and edges (relations). To exploit the graph-structured data which works on a non-Euclidean domain, several recent works propose graph-based neural architectures, referred to as *Graph Neural Networks (GNNs)* [14, 6]. While early works mostly deal with simple graphs with unlabeled edges, recently proposed relation-aware GNNs [3, 15] consider multi-relational graphs with labels and directions on the edges. These multi-relational graphs expand the application of GNNs to more real-world domains such as natural language understanding [16], modeling protein structure [17], drug-drug interaction prediction [18], retrosynthesis planning [19], to name a few.

Among multi-relational graphs, *Knowledge Graphs (KGs)*, which represent knowledge bases (KBs) such as Freebase [20] and WordNet [21], receive the most attention. They represent entities as nodes and relations among the entities as edges, in the form of a triplet: *(head entity, relation, tail entity)* (e.g., *(Louvre museum, is located in, Paris)*). Although knowledge graphs in general contain a huge amount of triplets, they are well known to be highly incomplete [22]. Therefore, automatically completing knowledge graphs, which is known as the *link prediction* task, is a practically important problem for KGs. Prior works tackle this problem, i.e., inferring missing triplets, by learning embeddings of entities and relations from existing triplets, and achieve impressive performances [23, 24, 25, 26, 27].

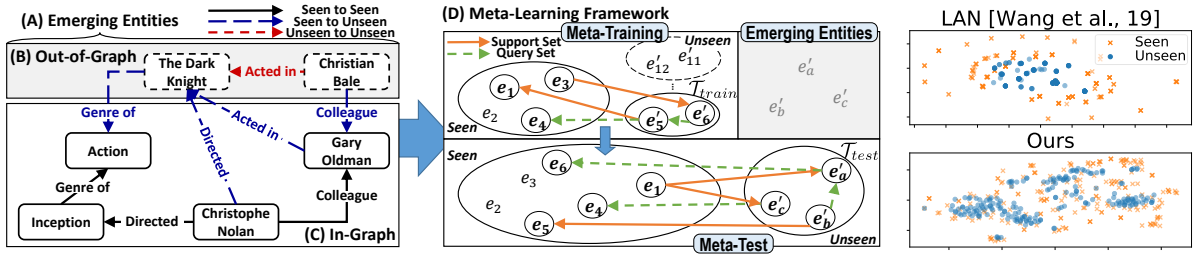

Figure 2.1: **Concept (Left):** An illustration of Out-of-Graph link prediction for emerging entities. Blue dotted arrows denote inferred relationships between seen and unseen entities, and red dotted arrows denote inferred relationships between unseen entities. **(Center):** An illustration of our meta-learning framework for the Out-of-Graph link prediction task. Orange arrows denote the support (training) set and green dotted arrows denote the query (test) set. **Visualization of embeddings (Right):** Our transductive GEN embeds the unseen entities on the manifold of seen entities, while the baseline [28] embeds the unseen entities off the manifold.

Despite such success, the link prediction for KGs in real-world scenarios remains challenging for a couple of reasons. First, knowledge graphs dynamically evolve over time, rather than staying static. Shi and Weninger [29] report that around 200 new entities emerge every day. Predicting links on these emerging entities pose a new challenge, especially when predicting the links between emerging (unseen) entities themselves. Moreover, real-world KGs generally exhibit long-tail distributions, where a large portion of the entities have only few triplets (See Figure 2.2). The embedding-based methods, however, usually assume that a sufficient number of associative triplets exist for training, and cannot embed unseen entities. Thus they are highly suboptimal for learning and inference on evolving real-world graphs.

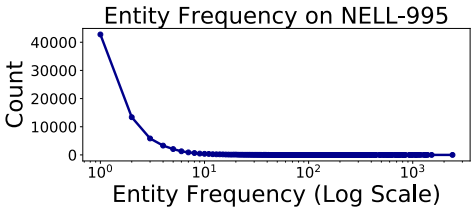

Figure 2.2: Entity frequency distribution of the NELL-995 knowledge graph dataset.

Motivated by the limitations of existing approaches, we introduce a realistic problem of *Few-Shot Out-of-Graph* (OOG) link prediction for emerging entities. In this task, we not only predict the links between seen and unseen entities but also between the *unseen entities* themselves (Figure 2.1, left). To this end, we propose a novel meta-learning framework for OOG link prediction, which we refer to as *Graph Extrapolation Networks (GENs)* (Figure 2.1, center). GENs are meta-learned to extrapolate the knowledge from seen to unseen entities, and transfer knowledge from entities with many to few links.

Specifically, given embeddings of the seen entities for a multi-relational graph, we meta-train two GNNs to predict the links between seen-to-unseen, and unseen-to-unseen entities. The first GNN, *inductive GEN*, learns to embed the unseen entities that are not observed, and predicts the links between seen and unseen entities. The second GNN, *transductive GEN*, learns to predict the links not only between seen and unseen entities, but also between unseen entities themselves. This transductive inference is possible since our meta-learning framework can simulate the unseen entities during meta-training, while they are unobservable in conventional learning schemes. Also, since link prediction for unseen entities is inherently unreliable, which gets worse when few triplets are available for each entity, we learn the distribution of unseen representations for stochastic embedding to account for the uncertainty. Further, we apply a transfer learning strategy to model the long-tail distribution. These lead GEN to represent the unseen entities that are well aligned with the seen entities (Figure 2.1, right).

We validate GENs for their OOG link prediction performance on three knowledge graph completion datasets, namely FB15K-237 [20], NELL-995 [30], and WN18RR [25]. We also validate GENs for OOG drug-drug interaction prediction task on DeepDDI [31] and BIOSNAP-sub [32] datasets. The experimental results on five datasets show that our model significantly outperforms the baselines, even when

they are retrained from scratch with unseen entities considered as seen entities. Further analysis of each component shows that both inductive and transductive layers of GEN help with the accurate link prediction for OOG entities. In sum, our main contributions are summarized as follows:

- We tackle a realistic problem setting of **few-shot out-of-graph link prediction**, aiming to perform link prediction not only between seen and unseen entities but also among unseen entities for multi-relational graphs that exhibit long-tail distributions, where each entity has only few triplets.
- To tackle this problem, we propose a **novel meta-learning framework**, Graph Extrapolation Network (GEN), which meta-learns the node embeddings for unseen entities, to obtain low error on link prediction for both seen-to-unseen (inductive) and unseen-to-unseen (transductive) cases.
- We validate GEN for few-shot out-of-graph link prediction tasks on **five benchmark datasets** for **knowledge graph completion** and **drug-drug interaction prediction**, on which it significantly outperforms relevant baselines, even when they are retrained with the unseen entities.

2.2 Related Work

Graph Neural Network Existing Graph Neural Networks (GNNs) encode the nodes by aggregating the features from the neighboring nodes, that use recurrent neural networks [33, 34], mean pooling with layer-wise propagation rules [6, 35], learnable attention-weighted combinations of the features [36, 37], to name a few. While most of the existing models work with simple undirected graphs, some recently proposed models tackle the multi-relational graphs for their practical importance. Directed-GCN [16] and Weighted-GCN [15] consider direction and relation types, respectively. Also, R-GCN [3] simultaneously considers direction and relation types. Similarly, MPNN [38] uses the edge-conditioned convolution to reflect the information on the edge types between nodes. Recently, Vashishth et al. [39] propose to jointly embed nodes and relations in a multi-relational graph. Since our GEN is a general framework for out-of-graph link prediction rather than a specific GNN architecture, it is compatible with any GNN implementations for multi-relational graphs.

Meta Learning Meta-learning, whose objective is to generalize over the distribution of tasks, is an essential approach for our few-shot out-of-graph link prediction framework, where we simulate the unseen nodes with a subset of training nodes. To mention a few, metric-based approaches [40, 41] learn a shared metric space to minimize the distance between correct and instance embeddings. On the other hand, gradient-based approaches [42, 43] learn shared parameters for initialization, to generalize over diverse tasks in a bi-level optimization framework. A few recent works consider meta-learning with GNNs, such as Satorras and Estrach [44] and Liu et al. [45] propose to meta-learn the GNNs for few-shot image classification, and Zhou et al. [46], Ding et al. [47] and Lan et al. [48] propose to meta-learn the GNNs for few-shot node classification. Further, Meta-Graph [49] proposes to construct graphs over the seen nodes, with only a small sample of known unlabeled edges.

Multi-relational Graph A popular application of multi-relation graphs is Knowledge Graph (KG) completion. Previous methods for this problem can be broadly classified as translational distance based [23, 50], semantic matching based [24, 51], convolutional neural network based [26, 25], and graph neural network based methods [3, 27]. While they require a large number of training instances to embed nodes and edges in a graph, many real-world graphs exhibit long-tail distributions. Few-shot relational

learning methods tackle this issue by learning few relations of seen entities [52, 53, 54]. Nonetheless, the problem becomes more difficult as real-world graphs have an evolving nature with new emerging entities. Several models [55, 56] tackle this problem by utilizing extra information about the entities, such as their textual description. Furthermore, some recent methods [57, 28, 58] propose to handle unseen entities in an inductive manner, to generate embeddings for unseen entities without re-training the entire model from scratch. However, since they can not simulate the unseen entities in the training phase, there are some fundamental limitations on the generalization for handling actual unseen entities. On the other hand, our method entirely tackles both of seen-to-unseen and unseen-to-unseen link prediction, under the transductive meta-learning framework that simulates the unseen entities during training. Drug-Drug Interaction (DDI) prediction is another important real-world application of multi-relational graphs, where the problem is to predict interactions between drugs. Recently, Zitnik et al. [18] and Ma et al. [59] propose end-to-end GNNs to tackle this problem, which demonstrate comparatively better performances over non-GNN methods [60, 61, 62].

2.3 Few-Shot Out-of-Graph Link Prediction

Our goal is to perform link prediction for emerging entities of multi-relational graphs, in which a large portion of the entities have only few triplets associated with them. We begin with the definitions of the multi-relational graph and the link prediction task, which we formalize in Definition 2.3.1 and Definition 2.3.2 as follows:

Definition 2.3.1: Multi-relational Graph

Let $\mathcal{E}$ and $\mathcal{R}$ be two sets of entities and relations respectively. Then a link is defined as a triplet (e_h, r, e_t), where $e_h, e_t \in \mathcal{E}$ are the head and the tail entity, and $r \in \mathcal{R}$ is a specific type of relation between the head and tail entities. A multi-relational graph $\mathcal{G}$ is represented as a collection of triplets. That is denoted as follows: $\mathcal{G} = \{(e_h, r, e_t)\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$.

Definition 2.3.2: Link Prediction

Link prediction refers to the task of predicting an unknown item of a triplet, when given two other items. We consider both of the entity prediction and relation prediction tasks. Entity prediction refers to the problem of predicting an unknown entity $e \subseteq \mathcal{E}$, given the entity and the relation: $(e_h, r, ?)$ or $(?, r, e_t)$. Relation prediction refers to the problem of predicting an unknown relation $r \subseteq \mathcal{R}$, given the head and tail entities: $(e_h, ?, e_t)$.

Link prediction for multi-relational graphs *Link prediction* is essentially the problem of assigning high scores to the true triplets, and therefore, many existing methods use score function $s(e_h, r, e_t)$ to measure the score of a given triplet, where the inputs depend on their respective embeddings (see Table 2.1). As a result, the objective of the link prediction is to find the representation of triplet elements and the function parameters in a parametric model case, which maximize the score of the true triplets. Which embedding methods to use depends on their specific application domains. However, existing works mostly tackle the link prediction

Table 2.1: Score functions for multi-relational graphs, where $\oplus$ denotes concatenation.

Model	Score Function	Domain
TransE [23]	$-\|\boldsymbol{e}_h + \boldsymbol{r} - \boldsymbol{e}_t\|_2$	Knowledge Graph
DistMult [24]	$\langle \boldsymbol{e}_h, \boldsymbol{r}, \boldsymbol{e}_t \rangle$	Knowledge Graph
Linear [38]	$\boldsymbol{r}(\boldsymbol{e}_h \oplus \boldsymbol{e}_t)$	Drug Interaction

between *seen* entities that already exist in the given multi-relational graph. In this work, we tackle a task of *few-shot Out-of-Graph (OOG)* link prediction formally defined in Definition 2.3.3 as follows:

Definition 2.3.3: Few-Shot Out-of-Graph Link Prediction

Given a graph $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$*, an unseen entity is an entity* $e' \in \mathcal{E}'$*, where* $\mathcal{E} \cap \mathcal{E}' = \emptyset$*. Then, out-of-graph link prediction is the problem of performing link prediction on* $(e', r, ?)$*,* $(?, r, e')$*,* $(e', ?, \tilde{e})$*, or* $(\tilde{e}, ?, e')$*, where* $\tilde{e} \in (\mathcal{E} \cup \mathcal{E}')$*. We further assume that each unseen entity* e' *is associated with* K *triplets:* $|\{(e', r, \tilde{e}) \text{ or } (\tilde{e}, r, e')\}| \leq K$ and $\tilde{e} \in (\mathcal{E} \cup \mathcal{E}')$*, where* K *is a small number (e.g.,* $K \leq 3$*).*

While few existing works [57, 28, 58] tackle the entity prediction between seen and unseen entities, in real-world settings, unseen entities do not emerge one by one but may emerge simultaneously as a set, with only few triplets available for each entity. Thus, they are highly suboptimal in handling such real-world scenarios, such as few-shot out-of-graph link prediction which we tackle in this work.

2.4 Learning to Extrapolate Knowledge with Graph Extrapolation Networks

We now introduce *Graph Extrapolation Networks* (GENs) for the out-of-graph (OOG) link prediction task. Since most of the previous methods assume that every entity in the test set is *seen* during training, they cannot handle *emerging* entities, which are unobserved during training. While few existing works [57, 28, 58] train for seen-to-seen link prediction with the hope that the models generalize on seen-to-unseen cases, they are suboptimal in handling unseen entities. Therefore, we use the *meta-learning* framework to handle the OOG link prediction problem, whose goal is to train a model over a distribution of tasks such that the model generalizes well on unseen tasks. Figure 2.1 illustrates our learning framework. Basically, we meta-train GEN which performs both inductive and transductive inference on various simulated test sets of OOG entities, such that it *extrapolates* the knowledge of existing graphs to any unseen entities. We describe the framework in detail in next few paragraphs.

Learning Objective Suppose that we are given a multi-relational graph $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, which consists of seen entities $e \in \mathcal{E}$ and relations $r \in \mathcal{R}$. Then, we aim to represent the unseen entities $e' \in \mathcal{E}'$ over a distribution $p(\mathcal{E}')$, by extrapolating the knowledge on a given graph $\mathcal{G}$, to predict the links between seen and unseen entities: (e, r, e') or (e', r, e), or even between unseen entities themselves: (e', r, e'). Toward this goal, we have to maximize the score of a true triplet $s(e_h, r, e_t)$ that contains any unseen entities e' to rank it higher than all the other false triplets, with embedding and score function parameters θ denoted as follows:

$$\max_{\theta} \mathbb{E}_{e' \sim p(\mathcal{E}')} \left[s(e', r, \tilde{e}; \theta) \text{ or } s(\tilde{e}, r, e'; \theta) \right], \quad \text{where} \quad \tilde{e} \in (\mathcal{E} \cup \mathcal{E}') \text{ and } e' \in \mathcal{E}'. \tag{2.1}$$

While this is a seemingly impossible goal as it involves generalization to real unseen entities, we can tackle it with meta-learning by simulating unseen entities during training, which we describe next.

Meta-Learning Framework While conventional learning frameworks can not handle unseen entities in the training phase, with meta-learning, we can formulate a set of tasks such that the model learns

Algorithm 1 Meta-Learning of GEN

Require: Distribution over training tasks $p(\mathcal{T}_{train})$
Require: Learning rate for meta-update α
Initialize parameters $\Theta = \{\theta, \theta_\mu, \theta_\sigma\}$
while not done **do**
  Sample a task $\mathcal{T} \sim p(\mathcal{T}_{train})$
  for all $e'_i \in \mathcal{T}$ **do**
    Sample support and query sets $\{\mathcal{S}_i, \mathcal{Q}_i\}$ correspond to e'_i
    Inductively generate using (2.3): $\phi_i = f_\theta(\mathcal{S}_i)$
  end for
  for all $e'_i \in \mathcal{T}$ **do**
    Transductively generate using (2.4): $\mu_i = g_{\theta_\mu}(\mathcal{S}_i, \phi)$ and $\sigma_i = g_{\theta_\sigma}(\mathcal{S}_i, \phi)$
    Sample $\phi'_i \sim \mathcal{N}\left(\mu_i, \text{diag}\left(\sigma_i^2\right)\right)$
  end for
  Update $\Theta \leftarrow \Theta - \alpha \nabla_\Theta \sum_i \mathcal{L}(\mathcal{Q}_i; \phi'_i)$ using (2.6)
end while

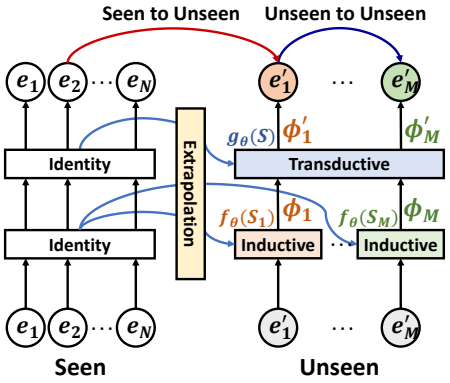

Figure 2.3: The overall framework of our model for each task. We extrapolate knowledge by using a support set S with inductive and transductive learning, and then predict links with the output embedding ϕ'.

to generalize over unseen entities, which are simulated using seen entities. To formulate the OOG link prediction problem into a meta-learning problem, we first randomly split the entities in a given graph into the meta-training set for simulated unseen entities, and the meta-test set for real unseen entities. Then, we generate a task by sampling the set of simulated unseen entities during meta-training, for the learned model to generalize over actual unseen entities (See Figure 2.1, center).

Formally, each task $\mathcal{T}$ over a distribution $p(\mathcal{T})$ corresponds to a set of unseen entities $\mathcal{E}_\mathcal{T} \subset \mathcal{E}'$, with a predefined number of instances $|\mathcal{E}_\mathcal{T}| = N$. Then we divide the triplets associative with each entity $e'_i \in \mathcal{E}_\mathcal{T}$ into the support set $\mathcal{S}_i$ and the query set $\mathcal{Q}_i$: $\mathcal{T} = \bigcup_{i=1}^{N} \mathcal{S}_i \cup \mathcal{Q}_i$, where $\mathcal{S}_i = \{(e'_i, r_j, \tilde{e}_j) \text{ or } (\tilde{e}_j, r_j, e'_i)\}_{j=1}^{K}$ and $\mathcal{Q}_i = \{(e'_i, r_j, \tilde{e}_j) \text{ or } (\tilde{e}_j, r_j, e'_i)\}_{j=K+1}^{M_i}$; $\tilde{e}_j \in (\mathcal{E} \cup \mathcal{E}')$. K is the few-shot size, and M_i is the number of triplets associated with each unseen entity e'_i. Our meta-objective is then learning to represent the unseen entities as ϕ using a support set $\mathcal{S}$ with a meta-function f, to maximize the triplet score on a query set $\mathcal{Q}$ with a score function s as follows:

$$\max_\theta \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\frac{1}{N} \sum_{i=1}^{N} \frac{1}{|\mathcal{Q}_i|} \sum_{j=K+1}^{M_i} s(e'_i, r_j, \tilde{e}_j; \phi_i, \theta) \text{ or } s(\tilde{e}_j, r_j, e'_i; \phi_i, \theta) \right], \ \phi_i = f_\theta(\mathcal{S}_i). \tag{2.2}$$

We refer to this specific setting as *K-shot out-of-graph (OOG) link prediction* throughout this paper. Once the model is trained with the meta-training tasks $\mathcal{T}_{train}$, we can apply it to unseen meta-test tasks $\mathcal{T}_{test}$, whose set of entities is disjoint from $\mathcal{T}_{train}$, as shown in the center of Figure 2.1.

Graph Extrapolation Networks In order to extrapolate knowledge of a given graph $\mathcal{G}$ to an unseen entity e'_i through a support set $\mathcal{S}_i$, we propose a GNN-based meta-learner that outputs the representation of unseen entities. We formulate our meta-learner $f_\theta(\cdot)$ as follows (Figure 2.3-Inductive):

$$f_\theta(\mathcal{S}_i) = \frac{1}{K} \sum_{(r,e) \in n(\mathcal{S}_i)} \mathbf{W}_r C_{r,e}, \tag{2.3}$$

where $n(\cdot)$ is a set of neighboring entities and relations: $n(\mathcal{S}_i) = \{(r, e) \,|\, (e'_i, r, e) \text{ or } (e, r, e'_i) \in \mathcal{S}_i\}$. Further, K is a size of $n(\mathcal{S}_i)$, $\mathbf{W}_r \in \mathbb{R}^{d \times 2d}$ is a relation-specific transformation matrix that is meta-learned, and $C_{r,e} \in \mathbb{R}^{2d}$ is a concatenation of feature representations of the relation-entity pair. Since GEN is essentially a framework for OOG link prediction, it is compatible with any GNNs.

Transductive Meta-Learning of GENs The previously described *inductive* GEN constructs the representation of each unseen entity e'_i through a support set $\mathcal{S}_i$, and then performs link prediction on a query set $\mathcal{Q}_i$, independently. A major drawback of this inductive scheme is that it does not consider the relationships between *unseen* entities. However, to tackle unseen entities simultaneously as a set, one should consider not only the relationships between seen and unseen entities as with the inductive GEN, but also among unseen entities themselves. To tackle this issue, we extend the inductive GEN to further perform a transductive inference, which will allow knowledge to propagate between unseen entities (see Subsection 2.7.5 of the Appendix for further discussions on inductive and transductive GENs).

More specifically, we add one more GEN layer $g_\theta(\cdot)$, which is similar to the inductive meta-learner $f_\theta(\cdot)$, to consider inter-relationships between unseen entities (Figure 2.3-Transductive):

$$g_\theta\left(\mathcal{S}_i, \phi\right) = \frac{1}{K} \sum_{(r,e)\in n(\mathcal{S}_i)} \mathbf{W}'_r C_{r,e} + \mathbf{W}_0 \phi_i, \tag{2.4}$$

where $\mathbf{W}_0 \in \mathbb{R}^{d\times d}$ is a weight matrix for the self-connection to consider the embedding ϕ_i, which is updated by the previous inductive layer $f_\theta(\mathcal{S}_i)$. To leverage the knowledge of neighboring *unseen* entities, our transductive layer $g_\theta(\cdot)$ aggregates the representations across all the neighbors with a weight matrix $\mathbf{W}'_r \in \mathbb{R}^{d\times 2d}$, where neighbors can include the unseen entities with embeddings ϕ, rather than treating them as noises or ignoring them as zero vectors like a previous inductive scheme.

Stochastic Inference A naive transductive GEN generalizes to the unseen entities by simulating them with the seen entities during meta-training. However, due to the intrinsic unreliability of few-shot OOG link prediction with each entity having only few triplets, there could be high uncertainties on the representations of unseen entities. To model such uncertainties, we *stochastically* embed the unseen entities by learning the distribution over an unseen entity embedding ϕ'_i. To this end, we first assume that the true posterior distribution has a following form: $p(\phi'_i \mid \mathcal{S}_i, \phi)$. Since computation of the true posterior distribution is intractable, we approximate the posterior using $q\left(\phi'_i \mid \mathcal{S}_i, \phi\right) = \mathcal{N}\left(\phi'_i \mid \mu_i, \text{diag}\left(\sigma_i^2\right)\right)$, and then compute the mean and variance via two individual transductive GEN layers: $\mu_i = g_{\theta_\mu}\left(\mathcal{S}_i, \phi\right)$ and $\sigma_i = g_{\theta_\sigma}\left(\mathcal{S}_i, \phi\right)$, which modifies the GraphVAE [63] to our setting. The form to maximize the score function s is then defined as follows:

$$s\left(e_h, r, e_t\right) = \frac{1}{L} \sum_{l=1}^{L} s\left(e_h, r, e_t; \phi'^{(l)}, \theta\right), \quad \phi'^{(l)} \sim q(\phi' \mid \mathcal{S}, \phi). \tag{2.5}$$

where we set the MC sample size to $L = 1$ during meta-training for computational efficiency. Also, we perform MC approximation with a sufficiently large sample size (e.g. $L = 10$) at meta-test. We let the approximate posterior same as the prior to make the consistent pipeline at training and test (see Sohn et al. [64]). We also model the source of uncertainty on the output embedding of an unseen entity from the transductive GEN layer via Monte Carlo dropout [65]. Our final *GEN* is then trained for both the inductive and transductive steps with stochastic inference, as described in Algorithm 1.

Loss Function Each task $\mathcal{T}$ that corresponds to a set of unseen entities $\mathcal{E}_\mathcal{T} \subset \mathcal{E}'$ consists of a support set and a query set: $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\}$. During training, we represent the embeddings of unseen entities $e'_i \in \mathcal{E}_\mathcal{T}$ using the support set $\mathcal{S}$ with GENs. After that, at the test time, we use the true labeled query set $\mathcal{Q}_i$ to optimize our GENs. Since every query set contains only positive triplets, we perform negative sampling [23, 24] to update a meta-learner by allowing it to distinguish positive from negative triplets.

Specifically, we replace the entity of each triplet in the query set: $\mathcal{Q}_i^- = \{(e'_i, r, e^-) \text{ or } (e^-, r, e'_i) \mid e^- \in \mathcal{E}\}$, where e^- is the corrupted entity. In this way, $\mathcal{Q}_i^-$ holds negative samples for an unseen entity e'_i. We then use hinge loss to optimize our model as follows:

$$\mathcal{L}(\mathcal{Q}_i) = \sum_{(e_h,r,e_t)\in\mathcal{Q}_i} \sum_{(e_h,r,e_t)^-\in\mathcal{Q}_i^-} \max\left\{\gamma - s^+(e_h,r,e_t) + s^-(e_h,r,e_t)^-, 0\right\}, \tag{2.6}$$

where $\gamma > 0$ is a margin hyper-parameter, and s is a specific score function in Table 2.1. s^+ and s^- denote the scores of positive and negative triplets, respectively. Notably, for the drug-drug interaction predict task, we follow Ryu et al. [31] to optimize our model, where binary cross-entropy loss is calculated for each label, with a sigmoid output of the linear score function in Table 2.1.

Meta-Learning for Long-Tail Tasks Since many real-world graphs follow the long-tail distributions (See Figure 2.2), it would be beneficial to transfer the knowledge from entities with many links to entities with few links. To this end, we follow a transfer learning scheme similar to Wang et al. [66]. Specifically, we start to learn the model with many shot cases, and then gradually decrease the number of shots to few shot cases in a logarithmic scale (see Subsection 2.7.2 of the Appendix for details).

2.5 Experiment

We validate GENs on few-shot out-of-graph (OOG) link prediction for two different domains of multi-relational graphs: knowledge graph (KG) completion and drug-drug interaction (DDI) prediction.

2.5.1 Knowledge Graph Completion

Datasets For knowledge graph completion datasets, we consider OOG *entity prediction*, whose goal is to predict the other entity given an unseen entity and a relation. **1) FB15k-237.** This dataset [67] consists of 310,116 triplets from 14,541 entities and 237 relations, which is collected via crowdsourcing. **2) NELL-995.** This dataset [30] consists of 154,213 triplets from 75,492 entities and 200 relations, which is collected by a lifelong learning system [68]. Since existing benchmark datasets do not target OOG link prediction, they assume that all entities given at the test time are seen during training. Therefore, we modify these two datasets such that the triplets used for link prediction at the test time contain at least one unseen entity (see Appendix 2.7.1 for the detailed dataset modification setup). **3) WN18RR.** This dataset [25] consists of 93,003 triplets from 40,943 entities and 11 relations extracted from WordNet [21]. Particularly, this dataset includes the unseen entities in validation and test sets, which overlaps with the 16 triplets to evaluate on a query set during meta-test. Therefore, we compare models only on these 16 triplets. Detailed descriptions of each dataset are reported in the Appendix 2.7.1.

Baselines and our models **1) TransE, 2) RotatE.** Translation distance based embedding methods for multi-relational graphs [23, 69]. **3) DistMult, 4) ComplEx.** Semantic matching based embedding methods [24, 51]. **5) R-GCN.** GNN-based method for modeling multi-relational data [3]. **6) MEAN, 7) LAN.** GNN-based methods for a out-of-knowledge base task, which tackle unseen entities without meta-learning [57, 28]. **8) GMatching, 9) MetaR, 10) FSRL.** Link prediction methods for unseen relations of *seen entities*, which we further train with our meta-learning framework [52, 53, 54]. **11) I-GEN.** An inductive version of our GEN which is meta-learned to embed an unseen entity. **12) T-GEN.**

Table 2.2: The results of 1- and 3-shot OOG link prediction on FB15k-237 and NELL-995. * means training a model within our meta-learning framework. Bold numbers denote the best results.

	Model	FB15k-237								NELL-995							
		MRR		H@1		H@3		H@10		MRR		H@1		H@3		H@10	
		1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S
Seen to Seen	TransE [23]	.053	.048	.034	.026	.050	.050	.082	.077	.009	.010	.002	.002	.007	.008	.020	.021
	DistMult [24]	.017	.014	.010	.009	.019	.014	.029	.022	.017	.016	.009	.008	.017	.017	.029	.028
	R-GCN [3]	.008	.006	.004	.003	.007	.005	.011	.010	.004	.004	.001	.001	.003	.003	.007	.006
Seen to Seen (with Support Set)	TransE [23]	.071	.120	.023	.057	.086	.137	.159	.238	.071	.118	.037	.061	.079	.132	.129	.223
	DistMult [24]	.059	.094	.034	.053	.064	.101	.103	.172	.075	.134	.045	.083	.083	.143	.131	.233
	ComplEx [51]	.062	.104	.037	.058	.067	.114	.110	.188	.069	.124	.045	.077	.071	.134	.117	.213
	RotatE [69]	.063	.115	.039	.069	.071	.131	.105	.200	.054	.112	.028	.060	.064	.131	.104	.209
	R-GCN [3]	.099	.140	.056	.082	.104	.154	.181	.255	.112	.199	.074	.141	.119	.219	.184	.307
Seen to Unseen	MEAN [57]	.105	.114	.052	.058	.109	.119	.207	.217	.158	.180	.107	.124	.173	.189	.263	.296
	LAN [28]	.112	.112	.057	.055	.118	.119	.214	.218	.159	.172	.111	.116	.172	.181	.255	.286
	GMatching [52]	.093	.105	.061	.061	.100	.112	.146	.183	.060	.079	.051	.059	.063	.097	.076	.106
	MetaR [53]	.076	.084	.043	.041	.084	.093	.133	.164	.092	.096	.059	.060	.107	.115	.154	.166
	FSRL [54]	.097	.090	.065	.058	.104	.096	.156	.150	.067	.085	.054	.064	.068	.095	.091	.126
Ours	GMatching* [52]	.224	.238	.157	.168	.249	.263	.352	.372	.120	.139	.074	.092	.136	.151	.215	.235
	MetaR* [53]	.294	.316	.223	.235	.318	.341	.441	.492	.177	.213	.104	.145	.217	.247	.315	.352
	FSRL* [54]	.255	.259	.187	.186	.279	.281	.391	.404	.130	.161	.075	.106	.145	.181	.253	.275
	I-GEN	.348	.367	.270	.281	.382	.407	.504	.537	.278	.285	.206	.214	.313	.322	.416	.426
	T-GEN	**.367**	**.382**	**.282**	**.289**	**.410**	**.430**	**.530**	**.565**	**.282**	**.291**	**.209**	**.217**	**.320**	**.333**	**.421**	**.433**

A transductive version of GEN, with additional stochastic transductive GNN layers to predict the link between unseen entities. We report detailed descriptions in the Appendix 2.7.1.

Implementation Details **1) Seen to Seen.** This scheme only trains seen-to-seen triplets from a meta-training set, without including unseen entities on a meta-test set. **2) Seen to Seen (with Support Set).** Following Xiong et al. [52], this scheme trains seen-to-seen link prediction baselines including support triplets of meta-validation and meta-test sets with unseen entities, since baselines are unable to solve the completely unseen entities at the test time. **3) Seen to Unseen.** This scheme tackles the link prediction for unseen entities without meta-learning [57, 28], or link prediction for unseen relations of seen entities with meta-learning [52, 53, 54]. **4) Ours.** Our meta-learning framework trains models only with a meta-training set, where we generate OOG entities using the episodic training [41]. For both I-GEN and T-GEN, we use **DistMult** for the initial embeddings of entities and relations, and the score function. We report detailed experimental setups in the Appendix 2.7.1.

Evaluation Metrics For evaluation, we use the ranking procedure by Bordes et al. [70]. For a triplet with an *unseen* head entity, we replace its corresponding tail entity with candidate entities from the dictionary to construct corrupted triplets. Then, we rank all the triplets, including the correct and corrupted ones by a scoring measure, to obtain a rank of the correct triplet. We provide the results using mean reciprocal rank (**MRR**) and Hits at n (**H@n**). Moreover, as done in previous works [23, 3], we measure the ranks in a filtered setting, where we do not consider triplets that appeared in either training, validation, or test sets. Finally, for a fair evaluation [71], we validate our models on different evaluation protocols, across which performances of our models remain consistent.

Main Results Table 2.2 shows that our I- and T-GEN outperform all baselines by impressive margins. Baseline models work poorly on emerging entities, even when they have seen the entities during training (with Support Set in Table 2.2). However, in our meta-learning framework, our GENs show superior performances over the baselines, even with a 1-shot setting. Moreover, while unseen relation prediction

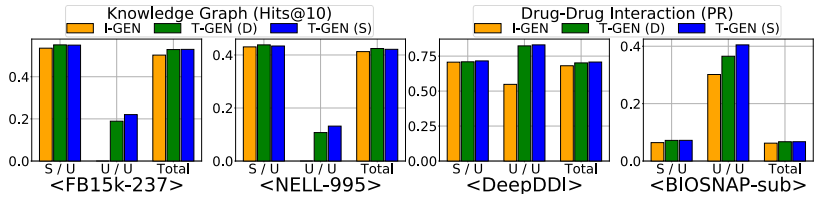

Figure 2.4: The results of seen to unseen (S/U), unseen to unseen (U/U) and total link prediction of I- and T-GEN with deterministic (D) and stochastic (S) modeling on KG completion and DDI prediction tasks.

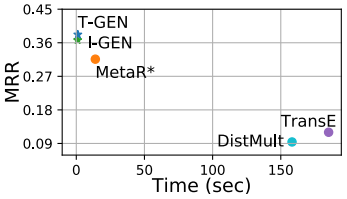

Figure 2.5: 3-shot OOG link prediction results, reported with MRR over training time.

baselines achieve extremely low performances compared to our GENs, we train baselines in our meta-learning framework and obtain significantly improved results. However, their performances are still substantially lower than GENs, which shows that GEN's dedicated embedding layers for seen-to-unseen and unseen-to-unseen link prediction are more effective for OOG link prediction.

Analysis on Seen to Unseen and Unseen to Unseen We observe that T-GEN outperforms I-GEN on both datasets by all evaluation metrics in Table 2.2. To see where the performance improvement comes from, we further examine the link prediction results for seen-to-unseen and unseen-to-unseen cases. Figure 2.4 shows that T-GEN obtains significant performance gain on the unseen-to-unseen link prediction problems, whereas I-GEN mostly cannot handle the unseen-to-unseen case as it does not consider the relationships between unseen nodes. Further, T-GEN with stochastic inference obtains even higher unseen-to-unseen link prediction performances, over deterministic T-GEN, which shows that modeling uncertainty in the latent embedding space of the unseen entities is effective.

Efficiency of Meta-Learning To demonstrate the efficiency of our meta-learning framework that embeds unseen entities without additional re-training, we compare GENs against models trained from scratch including unseen entities, for 3-shot OOG link prediction on FB15k-237. Figure 2.5 shows that GENs largely outperform baselines with a fraction of time required to embed unseen entities. Also, MetaR trained in our meta-learning framework is slower since it uses additional gradient information. This shows that GENs are efficient and generalize well to unseen entities with effective GNN layers.

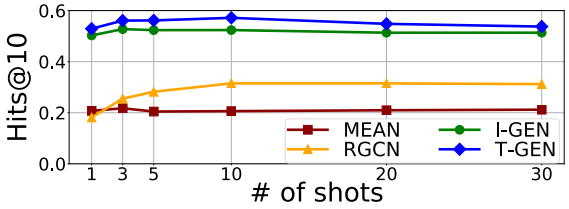

Figure 2.6: Diverse shots link prediction results with baselines and GENs on KG completion tasks.

	(Training) 1-Shot			(Training) 3-Shot		
Test	**MRR**	**H@1**	**H@10**	**MRR**	**H@1**	**H@10**
1-S	.367	.282	.530	.346	.262	.507
3-S	**.377**	**.288**	.556	**.382**	**.289**	.565
5-S	.362	.266	**.562**	.370	.269	**.570**
R-S	.375	.287	.548	.373	.282	.547

Table 2.3: Cross-shot learning results of T-GEN on KG completion tasks, by varying training and test shots.

Robustness on Many Shots While we mostly target a long-tail graph with the majority of entities having few links, our method works well on many-shot cases as well (Figure 2.6), on which GENs still largely outperform the baselines, even though R-GCN sees the unseen entities during training.

Robustness on Varying Shots We experiment our GEN with varying the number of shots by considering 1-, 3-, 5-, and random-shot (R-S: between 1 and 5) during meta-training and meta-test. Table 2.3 shows that differences in the number of shots used for training and test does not significantly affect the performance, which demonstrates the robustness of GENs on varying shots at test time. Moreover, our model trained on a 1-shot setting obtains even better performance on a 3-shot setting.

Model	WN18RR		
	MRR	**H@1**	**H@10**
DistMult [24]	.000	.000	.000
TransE [23]	.011	.000	.031
MetaR* [53]	.066	.063	.063
I-GEN	**.125**	**.125**	**.125**

Table 2.4: 1-shot OOG link prediction results on WN18RR for unseen entities.

Model	SI	Seen to Unseen		Unseen to Unseen	
		MRR	**H@3**	**MRR**	**H@3**
T-GEN	O	.379	.424	**.185**	**.187**
w/o transfer strategy	O	.374	.414	.183	.175
w/o pretrain	O	.361	.400	.168	.164
w/o stochastic inference	X	**.384**	**.425**	.153	.158
w/o transductive scheme	X	.366	.403	.000	.000

Table 2.5: Ablation study of T-GEN on FB15k-237. **SI** means whether to apply stochastic inference.

Results on Unseen Entities for WN18RR As previously mentioned, WN18RR dataset includes a small number of unseen entities in the test set. Therefore, we validate GEN only against test triplets that contain unseen entities in the test set. Table 2.4 shows that our GEN can improve the performance of out-of-graph link prediction even on this benchmark dataset.

Ablation Study We conduct an ablation study of the T-GEN on seen-to-unseen and unseen-to-unseen cases. Table 2.5 shows that using stochastic inference on the transductive layer helps significantly improve the unseen-to-unseen link prediction performance. Moreover, the meta-learning strategy of learning on entities with many links and then progressing to entities with few links performs well. Finally, we observe that using pre-trained embedding of a seen graph leads to better performance.

Qualitative Results We visualize the output representations of unseen entities with seen entities. Figure 2.1 (Right) shows that the embeddings of unseen entities are well aligned with the seen entities. Regarding concrete examples of the link prediction on NELL-995, see Subsection 2.7.4 of the Appendix.

2.5.2 Drug-Drug Interaction

Datasets We further validate our GENs on the OOG *relation prediction* task using two public Drug-Drug Interaction (DDI) datasets. **1) DeepDDI.** This dataset [31] consists of 1,861 drugs (entities) and 222,127 drug-drug pairs (triplets) from DrugBank [72], where 113 different relation types are used as labels. **2) BIOSNAP-sub.** This dataset [32, 59] consists of 645 drugs (entities) and 46,221 drug-drug pairs (triplets), where 200 different relation types are used as labels. Similar to the experiments on OOG knowledge graph completion tasks, we modify drug-drug interaction datasets for the OOG link prediction task. We report the detailed setup in Appendix 2.7.1.

Baselines and our models **1) MLP.** Feed-forward neural network used in the DDI task [31]. **2) MPNN.** Graph Neural Network that uses edge-conditioned convolution operations [38]. **3) R-GCN.** The same model used in the entity prediction on the KG completion task [3]. **4) I-GEN.** Inductive GEN, which only uses a feature representation of an entity $\boldsymbol{e}_k$, instead of a relation-entity pair $(\boldsymbol{r}_k, \boldsymbol{e}_k)$. This is because the relation is the prediction target for the DDI tasks. **5) T-GEN.** Transductive GEN with an additional transductive stochastic layer for unseen-to-unseen relation prediction.

Implementation Details For both I-GEN and T-GEN, we use **MPNN** for the initial embeddings of entities with a linear score function in Table 2.1. To train baselines, we use the Seen to Seen (with Support Set) scheme as in the KG completion task, where support triplets of meta-validation and meta-test sets are included during training. We report detailed experimental settings in the Appendix 2.7.1.

Evaluation Metrics For evaluation, we use the area under the receiver operating characteristic curve (**ROC**), the area under the precision-recall curve (**PR**), and the classification accuracy (**Acc**).

Main Results Table 2.6 shows the Drug-Drug Interaction (DDI) prediction performances of the baselines and GENs. Note that the performances on BIOSNAP-sub are comparatively lower in comparison to DeepDDI, due to the use of the preprocessed input features, as suggested by Ryu et al. [31]. Similar to the KG completion tasks, both I- and T-GEN outperform all baselines by impressive margins in all evaluation metrics. These results demonstrate that our GENs can be easily extended to OOG link prediction for other real-world applications of multi-relational graphs.

Table 2.6: Results of 3-shot OOG *relation* prediction on DeepDDI and BIOSNAP-sub.

	DeepDDI			**BIOSNAP-sub**		
Model	**ROC**	**PR**	**Acc**	**ROC**	**PR**	**Acc**
MLP	.928	.476	.528	.597	.034	.049
MPNN [38]	.939	.478	.681	.597	.026	.067
R-GCN [3]	.928	.397	.640	.594	.041	.051
I-GEN	.946	.681	.807	.608	.062	.073
T-GEN	**.954**	**.708**	**.815**	**.625**	**.067**	**.089**

Analysis on Seen to Unseen and Unseen to Unseen We also compare the link prediction performance for both seen-to-unseen and unseen-to-unseen cases on two DDI datasets. The rightmost two columns of Figure 2.4 show that T-GEN obtains superior performance over I-GEN on unseen-to-unseen link prediction cases, especially when utilizing stochastic modeling schemes.

2.6 Conclusion

We formally defined a realistic problem of the *few-shot out-of-graph (OOG)* link prediction task, which considers link prediction not only between seen to unseen (or emerging) entities but also between unseen entities for multi-relational graphs, where each entity comes with only few associative triplets to train. To this end, we proposed a novel meta-learning framework for OOG link prediction, which we refer to as *Graph Extrapolation Network (GEN)*. Under the defined K-shot learning setting, GENs learn to extrapolate the knowledge of a given graph to unseen entities, with a stochastic transductive layer to further propagate the knowledge between the unseen entities and to model uncertainty in the link prediction. We validated the OOG link prediction performance of GENs on five benchmark datasets, on which proposed models largely outperformed the relevant baselines.

2.7 Appendix

2.7.1 Experimental Setup

Datasets Since existing benchmark datasets assume that all entities given at the test time are seen during training, we modify the datasets to formulate the Out-of-Graph (OOG) link prediction task, where completely unseen entities appear at the test time. Dataset modification processes are as follows:

- First, we randomly sample the unseen entities, which have a relatively small amount of triplets on each dataset. We then divide the sampled unseen entities into meta-training/validation/test sets.
- Second, we select the triplets which are used for constructing an In-Graph, where the head and tail entities of every triplet in the In-Graph consist of only seen entities, not any unseen entity.
- Finally, we match the unseen entities in the meta-sets with their triplets. Each triplet in meta-sets contains at least one unseen entity. Also, every triplet in meta-sets is not included in the In-Graph.

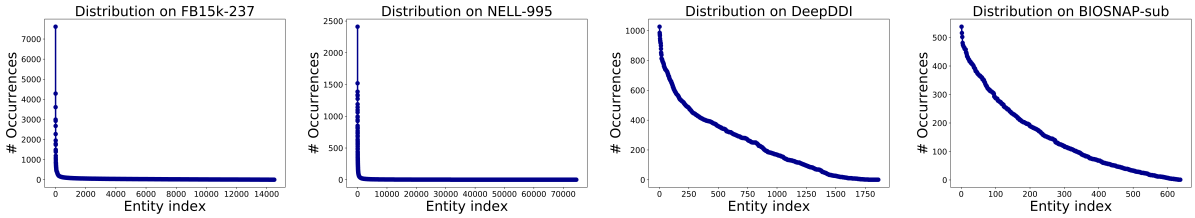

Figure 2.7: Distribution of entity occurrences on FB15k-237, NELL-995, DeepDDI, and BIOSNAP-sub.

1) FB15k-237. This dataset [67] consists of 14,541 entities and 237 relations, which is collected from crowdsourcing and used for the knowledge graph completion task. We randomly sample the 5,000 entities from 10,938 entities, which have associated triplets between 10 and 100. Also, we split the entities such that we have 2,500/1,000/1,500 unseen (Out-of-Graph) entities and 72,065/6,246/9,867 associated triplets containing unseen entities for meta-training/validation/test. The remaining triplets that do not hold an unseen entity are used for constructing an In-Graph. As shown in Figure 2.7, this dataset follows a highly long-tailed distribution.

2) NELL-995. This dataset [30] consists of 75,492 entities and 200 relations, which is collected by a lifelong learning system [68] and used for the knowledge graph completion task. We randomly sample the 3,000 entities from 5,694 entities, which have associated triplets between 7 and 100. Also, we split the entities such that we have 1,500/600/900 unseen (Out-of-Graph) entities and 22,345/3,676/5,852 associated triplets containing unseen entities for meta-training/validation/test. The remaining triplets that do not hold an unseen entity are used for constructing an In-Graph. As shown in Figure 2.7, this dataset follows a highly long-tailed distribution.

3) WN18RR. This dataset [25] consists of 93,003 triplets from 40,943 entities and 11 relations, which is collected from WordNet [21] and used for the knowledge graph completion task. Particularly, this dataset essentially contains 198 unseen entities over 210 triplets on the validation set and 209 unseen entities over 210 triplets on the test set.

Note that, to construct a support set for training and a query set for test in our meta-learning framework, we need at least two triplets for each unseen entity. Therefore, even in the WN18RR that contains an appropriate number of unseen entities, the amount of triplets to evaluate on a query set is too small (only 16 triplets to test, which is 0.02 % compared to the number of all triplets), in which we consider validation and test sets together since test set only has 3 triplets to test. In other words, most unseen entities on WN18RR have only one triplet, which reflects the long-tail distribution of real-world graphs for emerging entities. Thus, we compare models only on these 16 triplets during meta-test.

To use the meta-learning framework from the conventional learning scheme, we randomly sample the 3,000 unseen entities from 4,478 entities, which have associated triplets between 8 and 100. After that, we use the sampled 3,000 unseen entities for meta-training which has 36,166 overlapped triplets. The remaining triplets that do not hold an unseen entity are used for constructing an In-Graph.

4) DeepDDI. This dataset [31] consists of 1,861 entities and 113 relations, which is collected from the DrugBank database [72] and used for the drug-drug interaction prediction task. We randomly sample the 500 entities from 1,039 entities, which have associated triplets between 7 and 300. Also, we split the entities such that we have 250/100/150 unseen (Out-of-Graph) entities and 27,726/1,171/2,160 associated triplets containing unseen entities for meta-training/validation/test. The remaining triplets that do not hold an unseen entity are used for constructing an In-Graph.

5) BIOSNAP-sub. This dataset [59] consists of 637 entities and 200 relations, which is col-

lected from the BIOSNAP [32], further modified by Ma et al. [59] for efficiency and used for the drug-drug interaction prediction task. We randomly sample the 150 entities from 507 entities, which have associated triplets between 7 and 300. Also, we split the entities such that we have 75/30/45 unseen (Out-of-Graph) entities and 7,140/333/643 associated triplets containing unseen entities for meta-training/validation/test. The remaining triplets that do not hold an unseen entity are used for constructing an In-Graph.

Baselines and Our Models for Knowledge Graph Completion We describe the baseline models and our graph extrapolation networks for few-shot out-of-graph *entity prediction* on the knowledge graph (KG) completion task.

1) TransE. This is a translation embedding model for relational data by Bordes et al. [23]. It represents both entities and relations as vectors in the same space, where the relation in a triplet is used as a translation operation between the head and the tail entity.

2) RotatE. This model represents entities as complex vectors and relations as rotations in a complex vector space [69], which extends TransE with a complex operation.

3) DistMult. This model represents the relationship between the head and the tail entity in a bi-linear formulation, which can capture pairwise interaction between entities [24].

4) ComplEx. This model extends the DistMult by introducing embeddings on a complex space to consider asymmetric relations, where scores are measured based on the order of the entities [51].

5) R-GCN. This is a GNN-based method for modeling relational data, which extends the graph convolutional network to consider multi-relational structures, by Schlichtkrull et al. [3].

6) MEAN. This model computes the embedding of entities by GNN-based neighboring aggregation scheme, where they only train for seen-to-seen link prediction, with the hope that the model generalizes on seen-to-unseen cases, without meta-learning [57].

7) LAN. This model extends the MEAN [57] to consider relations with neighboring information by utilizing attention mechanisms, without meta-learning [28].

8) GMatching. This model tackles the link prediction on unseen relations of *seen entities* by searching for the closest entity pair with meta-learning [52]. We further extend it in our meta-learning framework such that it can handle unseen entities.

9) MetaR. This model tackles the link prediction on unseen relations of *seen entities* by generating the embeddings of unseen relations with gradient information over the meta-learning framework [53]. We further extend it in our meta-learning framework such that it can handle unseen entities.

10) FSRL. This model extends the GMatching [52] to tackle the link prediction on unseen relations of *seen entities* by utilizing attention mechanisms with meta-learning [54]. We further extend it in our meta-learning framework such that it can handle unseen entities.

11) I-GEN. This is an inductive version of our Graph Extrapolation Network (GEN), that is meta-learned to embed an unseen entity to infer hidden links between seen and unseen entities.

12) T-GEN. This is a transductive version of GEN, with additional stochastic transductive GNN layers on top of the I-GEN, that is meta-learned to predict the links between unseen entities as well as between seen and unseen entities.

Baselines and Our Models for Drug-Drug Interaction We describe the baseline models and our graph extrapolation networks for few-shot out-of-graph *relation prediction* on the drug-drug interaction (DDI) task.

1) MLP. This is a feed-forward neural network model used for DeepDDI [31] dataset. It classifies the relation of two drugs using their pairwise features.

2) MPNN. This is a GNN-based model that uses features for relation types with edge-conditioned convolution operations [38].

3) R-GCN. This is the same model used in the entity prediction on knowledge graph completion tasks [3], applied to drug-drug interaction tasks.

4) I-GEN. This is an inductive GEN, which only uses the feature representation of the entity $\boldsymbol{e}_k$ when aggregating neighboring information, instead of using the concatenated representation of the relation-entity pair $(\boldsymbol{r}_k, \boldsymbol{e}_k)$ like KG completion tasks. This is because the relation is the prediction target for the DDI tasks.

5) T-GEN. This is a transductive version of GEN, with additional transductive stochastic layers on top of the I-GEN, for unseen-to-unseen relation prediction as well as seen-to-unseen prediction.

Common Implementation Details For every dataset, we set the embedding dimension for both entity and relation as 100. Also, we set the embedding of unseen entities as the zero vector. Furthermore, since we consider a highly multi-relational graph, we use the basis decomposition on weight matrices $\mathbf{W}_r$ and $\mathbf{W}'_r$ to prevent the excessive increase in the model size, which is proposed in Schlichtkrull et al. [3]: $\mathbf{W}_r = \sum_{b=1}^{B} a_{r_b} \mathbf{V}_b$, where B is a number of basis, a_{r_b} is a coefficient of each relation $r \in \mathcal{R}$, and $\mathbf{V}_b \in \mathbb{R}^{d \times 2d}$ is a shared representation of various relations. For all experiments, we use PyTorch [73] and PyTorch geometric [74] frameworks, and train on a single Titan XP or a single GeForce RTX 2080 Ti GPU. We optimize the proposed GENs using Adam [75].

Implementation Details on Knowledge Graph Completion For both I-GEN and T-GEN, we search for the learning rate α in the range of $\{3 \times 10^{-4}, 1 \times 10^{-3}, 3 \times 10^{-3}\}$, margin γ in the range of $\{0.25, 0.5, 1\}$, and dropout ratio at every GEN layer in the range of $\{0.1, 0.2, 0.3\}$. As a score function, we use DistMult [24] at the end of our GENs. For all datasets, we consider the inverse relation as suggested by several recent works for multi-relational graphs [16, 3, 39], since directed relation information flows along with both directions. Finally, to select the best model, we use the mean reciprocal rank (MRR) as an evaluation metric.

For **FB15k-237** dataset, we set the $\alpha = 1 \times 10^{-3}$ and $\gamma = 1$ with dropout rate 0.3. Also, we set the number of basis units $B = 100$ for the basis decomposition on each GEN layer, and sample 32 negative triplets for each positive triplet in both I-GEN and T-GEN. At every episodic training, we randomly sample 500 unseen entities in the meta-training set. Also, we validate and test models using all unseen entities in the meta-validation and meta-test sets, respectively.

For **NELL-995** dataset, we use the same settings with FB15k-237, except that we sample 64 negative triplets for each positive triplet.

For **WN18RR** dataset, we use the same settings with FB15k-237, except that we randomly sample 100 unseen entities for episodic training during meta-training.

Implementation Details on Drug-Drug Interaction For both I-GEN and T-GEN, we search for the learning rate α in the range of $\{5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}\}$, and dropout ratio at every GEN layer in the range of $\{0.1, 0.2, 0.3\}$. As a score function, we use two linear layers with ReLU as an activation function at the end of the first layer. For all datasets, we consider the inverse relation as in the case of the knowledge graph completion task. Finally, to select the best model, we use the area under the

Table 2.7: The naive and meta-learning strategy results of 1- and 3-shot OOG link prediction on FB15k-237 and NELL-995. Bold numbers denote the best results on I-GEN and T-GEN, respectively.

Model	FB15k-237								NELL-995							
	MRR		H@1		H@3		H@10		MRR		H@1		H@3		H@10	
	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S
I-GEN	**.348**	**.367**	**.270**	**.281**	**.382**	**.407**	**.504**	**.537**	**.278**	**.285**	**.206**	**.214**	**.313**	**.322**	**.416**	**.426**
w/o transfer strategy	.344	.362	.264	.275	.379	.401	.503	.527	.272	.277	.198	.206	.309	.314	.413	.414
T-GEN	**.367**	**.382**	**.282**	.289	**.410**	**.430**	**.530**	**.565**	**.282**	**.291**	**.209**	**.217**	**.320**	**.333**	**.421**	**.433**
w/o transfer strategy	.362	.381	.278	**.291**	.400	.422	.527	.563	.273	.290	.198	**.217**	.310	.326	.412	.431

receiver operating characteristic curve (ROC) as an evaluation metric.

For **DeepDDI** dataset, we set the $\alpha = 1 \times 10^{-3}$ with dropout rate 0.3 for both I-GEN and T-GEN. Also, we set the number of basis units $B = 200$ for the basis decomposition. At every episodic training, we randomly sample 80 unseen entities in the meta-training set. Also, we validate and test models using all unseen entities in the meta-validation and meta-test sets, respectively.

For **BIOSNAP-sub** dataset, we set the $\alpha = 1 \times 10^{-3}$ with dropout rate 0.1 for I-GEN and 0.2 for T-GEN. Also, we set the number of basis units $B = 200$ for the basis decomposition. At every episodic training, we randomly sample 50 unseen entities in the meta-training set. Also, we validate and test models using all unseen entities in the meta-validation and meta-test sets, respectively.

2.7.2 Meta-learning for Long-tail Tasks

Implementation Details Many real-world graphs follow the long-tail distribution, where few entities have many links while the majority have few links (See Figure 2.7). For such an imbalanced graph, it would be beneficial to transfer the knowledge from entities with many links to entities with few links. To this end, we transfer the meta-knowledge on data-rich entities to data-poor entities by simulating the data-rich circumstance under the meta-learning framework, motivated by Wang et al. [66]. Specifically, we first meta-train our GENs with many shot cases (e.g. $K = 10$), and then gradually decrease the number of shots to few shots cases (e.g. $K = 1$ or 3) in logarithmic scale: $K_i = \lfloor \log_2(\text{max-iteration}/i) \rfloor + K$, where K_i is the training shot size at the current iteration number i, and K is the test shot size. In this way, GENs learn to represent the unseen entities using data-rich instances, and entities with few links regimes may experience like data-rich instances, with the model parameters trained on the entities with many links and tuned on the entities with few links.

More Ablation Studies Since knowledge graphs follow a highly long-tailed distribution (See Figure 2.7), we provide the more experimental results about transfer strategies on knowledge graph completion tasks, to demonstrate the effectiveness of the proposed meta-learning scheme on a long-tail task. Table 2.7 shows that the transfer strategy outperforms naive I-GEN and T-GEN on all evaluation metrics, except for only two H@1 cases of T-GEN on 3-shot OOG link prediction settings. We conjecture that the effectiveness of the meta-learning scheme is especially larger on 1-shot cases, where data is extremely poor, rather than the 3-shot cases.

2.7.3 Additional Experimental Results

Effect of Score Function While we performed all experiments with DistMult score function in the main paper, we further evaluate proposed GENs on the few-shot OOG link prediction task with

Table 2.8: Total, seen-to-unseen and unseen-to-unseen results of 1- and 3-shot OOG link prediction on FB15k-237. * means training a model within our meta-learning framework. Bold numbers denote the best results.

	Model	Total								Seen to Unseen				Unseen to Unseen			
		MRR		H@1		H@3		H@10		MRR		H@10		MRR		H@10	
		1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S	1-S	3-S
Seen to Seen	TransE [23]	.053	.048	.034	.026	.050	.050	.082	.077	.055	.050	.086	.081	.016	.014	.029	.025
	DistMult [24]	.017	.014	.010	.009	.019	.014	.029	.022	.018	.015	.029	.022	.011	.007	.025	.015
	R-GCN [3]	.008	.006	.004	.003	.007	.005	.011	.010	.003	.003	.005	.006	.076	.050	.101	.070
Seen to Unseen	MEAN [57]	.105	.114	.052	.058	.109	.119	.207	.217	.112	.121	.221	.231	.000	.000	.000	.000
	LAN [28]	.112	.112	.057	.055	.118	.119	.214	.218	.119	.119	.228	.232	.000	.000	.000	.000
	GMatching* [52]	.224	.238	.157	.168	.249	.263	.352	.372	.239	.254	.375	.400	.000	.000	.000	.000
Ours	I-GEN (Random)	.309	.319	.236	240	.337	.352	.455	.477	.329	.339	.485	.508	.000	.000	.000	.000
	I-GEN (DistMult)	.348	.367	.270	.281	.382	.407	.504	.537	.371	.391	.537	.571	.000	.000	.000	.000
	I-GEN (TransE)	.345	.371	.259	.275	.385	.416	.515	.559	.367	.395	.548	**.594**	.000	.000	.000	.000
	T-GEN (Random)	.349	.360	.268	.273	.385	.398	.508	.532	.361	.373	.529	.554	.168	.164	.185	.192
	T-GEN (DistMult)	**.367**	**.382**	**.282**	**.289**	**.410**	**.430**	.530	**.565**	**.379**	**.396**	.550	.588	**.185**	**.175**	**.220**	.201
	T-GEN (TransE)	.356	.374	.267	.282	.403	.425	**.531**	.552	.368	.387	**.552**	.572	.175	**.175**	.205	**.235**

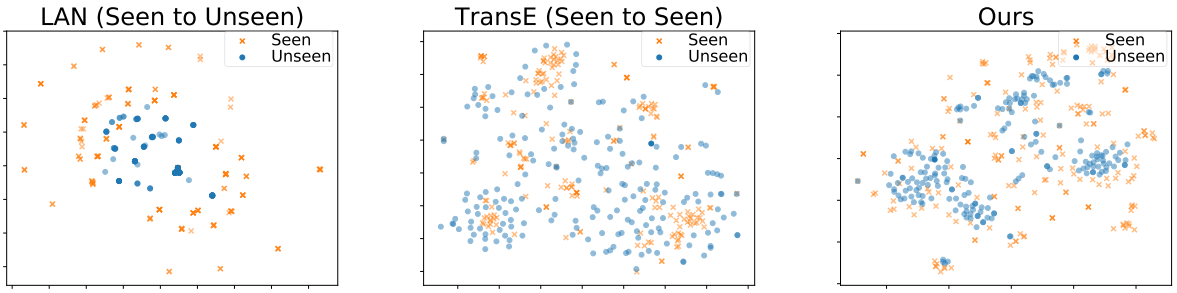

Figure 2.8: T-SNE visualization of the learned embeddings for seen and unseen entities.

TransE [23], which is another popular score function. We use the same settings as with DistMult [24] experiments, except that we use TransE for the initial embedding and the score measurement. Table 2.8 shows that our I-GEN and T-GEN with TransE score function also outperform all baselines by impressive margins, where they perform comparably to DistMult. These results suggest that our model works regardless of the score function.

Effect of Initialization We further demonstrate the meta-training effectiveness of our meta-learner, by randomly initializing an In-Graph, in which GEN extrapolates knowledge for an unseen entity without using the pre-trained embeddings of entity and relation. Table 2.8 shows that, while results with the random initialization are lower than pre-trained models, GENs are still powerful on the unseen entity, compared to the baselines. These results suggest that GENs trained under the meta-learning framework can be applied to more difficult situations, as pre-trained In-Graph might not be available for the few-shot OOG link prediction in real-world scenarios.

Effect of Transductive Scheme As shown in Table 2.8, I-GEN achieves comparable performances with T-GEN on seen-to-unseen link prediction. However, since the inductive method can not handle two unseen entities at once, this scheme does not solve the unseen-to-unseen link prediction for two emerging entities. However, unseen entities do not emerge one by one, but may emerge simultaneously as a set in real-world settings, such that we consider a transductive scheme to deal with this challenging circumstance. Table 2.8 shows that, while the unseen-to-unseen link prediction performances of T-GEN is far from the seen-to-unseen performances, T-GEN can infer hidden relationships among unseen entities by transductive learning and inference.

Table 2.9: Examples of OOG link prediction on NELL-995. S: seen, U: unseen, O: correct prediction, X: incorrect prediction, (H): head entity, (R): relation, (T): tail entity, and _: unseen entity.

Type	**I-GEN**	**T-GEN**	**Triplet**
S-U	O	O	(H) musician_vivaldi, (R) musician_plays_instrument, (T) music_instrument_string
S-U	O	O	(H) city_hawthorne, (R) city_located_in_state, (T) state_or_province_california
S-U	O	O	(H) journalist_maureen_dowd, (R) works_for, (T) company_york_times
S-U	O	O	(H) person_monroe, (R) person_born_in_location, (T) county_york_city
S-U	O	O	(H) ceo_stan_o_neal, (R) works_for, (T) retailstore_merrill
S-U	O	O	(H) insect_insects, (R) invertebrate_feed_on_food , (T) agricultural_product_wood
U-U	X	O	(H) person_katsuaki_watanabe, (R) person_leads_organization, (T) automobilemaker_toyota
U-U	X	O	(H) mlauthor_web_search, (R) agent_competes_with_agent, (T) website_altavista_com
U-U	X	O	(H) chemical_chromium, (R) chemical_is_type_of_chemical, (T) chemical_heavy_metals
U-U	X	X	(H) food_meals, (R) food_decreases_the_risk_of_disease, (T) disease_heart_disease

More Visualization The experimental results on multiple datasets show that our GENs significantly outperform baselines, even when they are retrained with the unseen entities. To see why does GENs generalize well to the unseen entities, we visualize the output embeddings of seen-to-unseen baseline (LAN), seen-to-seen baseline (TransE) which is retrained from scratch, and T-GEN. As shown in Figure 2.8, since GEN embeds the unseen entities on the manifold of seen entities, it achieves better results on few-shot OOG link prediction tasks than baseline models.

2.7.4 Examples

Table 2.9 shows some concrete examples of the OOG link prediction result from NELL-995 dataset, where the 7 to 9 rows show that our T-GEN correctly performs link prediction for two unseen entities.

2.7.5 Discussion on Inductive and Transductive Schemes

In this subsection, we describe in detail about task-level transductive inference and meta-level inductive inference for the proposed transductive GEN (T-GEN) model. Since transductive GEN requires to predict links between two *unseen test entities* which is impossible to handle using conventional link pre-

diction approaches, the problem is indeed transductive. Furthermore, the inference of unseen-to-unseen links could be also considered as inductive at meta-level, where we inductively learn the parameters of GEN across the batch of tasks. Thus, we are tackling transductive inference problems by considering them as meta-level inductive problems, but the intrinsic unseen-to-unseen link prediction is still transductive. To illustrate more concretely, different sets of unseen entities make mutually inconsistent predictions, which is caused by transduction. Other transductive meta-learning approaches such as TPN [76] and EGNN [77] tackle the problem with similar high-level ideas, where they classify unseen classes by leveraging both information on labeled and unlabeled nodes.

This Chapter is based on the work that is published at ICLR 2021 [78].

Chapter 3. Accurate Learning of Entire Graph Representations

How do we obtain a compact representation of an entire graph, often having a large number of nodes, while expressively distinguishing two different graphs? We summarize the formulation of our graph polling scheme, and the theoretical and empirical results that we obtain, in below.

Graph neural networks have been widely used on modeling graph data, achieving impressive results on node classification and link prediction tasks. Yet, obtaining an accurate representation for a graph further requires a pooling function that maps a set of node representations into a compact form. A simple sum or average over all node representations considers all node features equally without consideration of their task relevance, and any structural dependencies among them. Recently proposed hierarchical graph pooling methods, on the other hand, may yield the same representation for two different graphs that are distinguished by the Weisfeiler-Lehman test, as they suboptimally preserve information from the node features. To tackle these limitations of existing graph pooling methods, we first formulate the graph pooling problem as a multiset encoding problem with auxiliary information about the graph structure, and propose a *Graph Multiset Transformer* (GMT) which is a multi-head attention based global pooling layer that captures the interaction between nodes according to their structural dependencies. We show that GMT satisfies both injectiveness and permutation invariance, such that it is at most as powerful as the Weisfeiler-Lehman graph isomorphism test. Moreover, our methods can be easily extended to the previous node clustering approaches for hierarchical graph pooling. Our experimental results show that GMT significantly outperforms state-of-the-art graph pooling methods on graph classification benchmarks with high memory and time efficiency, and obtains even larger performance gain on graph reconstruction and generation tasks.

3.1 Introduction

Graph neural networks (GNNs) [4, 5], which work with graph structured data, have recently attracted considerable attention, as they can learn expressive representations for various graph-related tasks such as node classification, link prediction, and graph classification. While the majority of the existing works on GNNs focus on the message passing strategies for neighborhood aggregation [6, 35], which aims to encode the nodes in a graph accurately, graph pooling [79, 80] that maps the set of nodes into a compact representation is crucial in capturing a meaningful structure of an entire graph.

As a simplest approach for graph pooling, we can average or sum all node features in the given graph [81, 10] (Figure 3.1 (B)). However, since such simple aggregation schemes treat all nodes equally without considering their relative importance on the given tasks, they can not generate a meaningful graph representation in a task-specific manner. Their flat architecture designs also restrict their capability toward the hierarchical pooling or graph compression into few nodes. To tackle these limitations, several differentiable pooling operations have been proposed to condense the given graph. There are two dominant approaches to pooling a graph. Node drop methods [79, 82] (Figure 3.1 (C)) obtain a score of each node using information from graph convolutional layers, and then drop unnecessary nodes with

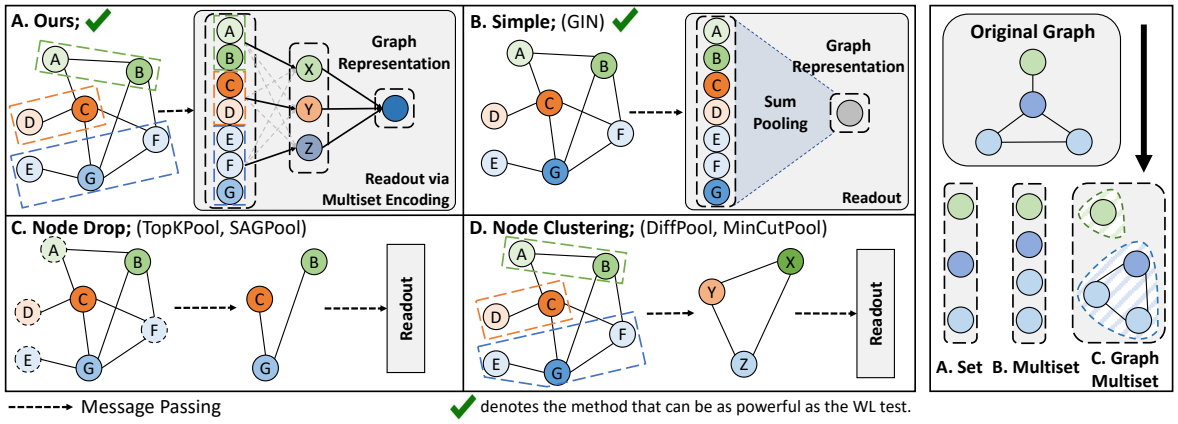

Figure 3.1: **Concepts (Left):** Conceptual comparison of graph pooling methods. Grey box indicates the readout layer, which is compatible with our method. Also, green check icon indicates the model that can be as powerful as the WL test. **(Right):** An illustration of set, multiset, and graph multiset encoding for graph representation.

lower scores at each pooling step. Node clustering methods [80, 83] (Figure 3.1 (D)), on the other hand, cluster similar nodes into a single node by exploiting their hierarchical structure.

Both graph pooling approaches have obvious drawbacks. First, node drop methods unnecessarily drop some nodes at every pooling step, leading to information loss on those discarded nodes. On the other hand, node clustering methods compute the *dense* cluster assignment matrix with an adjacency matrix. This prevents them from exploiting sparsity in the graph topology, leading to excessively high computational complexity [82]. Furthermore, to accurately represent the graph, the GNNs should obtain a representation that is as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test [12], such that it can map two different graphs onto two distinct embeddings. While recent message-passing operations satisfy this constraint [84, 10], most deep graph pooling works [80, 82, 85, 83] overlook graph isomorphism except for a few [79].

To obtain accurate representations of graphs, we need a graph pooling function that is as powerful as the WL test in distinguishing two different graphs. To this end, we first focus on that the graph representation learning can be regarded as a *multiset* encoding problem, which allows for possibly repeating elements, since a graph may have redundant node representations (See Figure 3.1, right). However, since a graph is more than a multiset due to its structural constraint, we further define the problem as a *graph multiset* encoding, whose goal is to encode two different graphs, given as multisets of node features with auxiliary structural dependencies among them (See Figure 3.1, right), into two unique embeddings. We tackle this problem by utilizing a graph-structured attention unit. By leveraging this unit as a fundamental building block, we propose the *Graph Multiset Transformer* (GMT), a pooling mechanism that condenses the given graph into the set of representative nodes, and then further encodes relationships between them to enhance the representation power of a graph. We theoretically analyze the connection between our pooling operations and WL test, and further show that our graph multiset pooling function can be easily extended to node clustering methods.

We then experimentally validate the **graph classification** performance of GMT on 10 benchmark datasets from biochemical and social domains, on which it significantly outperforms existing methods on most of them. However, since graph classification tasks only require discriminative information, to better quantify the amount of information about the graph in condensed nodes after pooling, we further validate it on **graph reconstruction** of synthetic and molecule graphs, and also on two **graph generation** tasks, namely molecule generation and retrosynthesis. Notably, GMT outperforms baselines with even larger

performance gap on graph reconstruction, which demonstrates that it learns meaningful information without forgetting original graph structure. Finally, it improves the graph generation performance on two tasks, which shows that GMT can be well coupled with other GNNs for graph representation learning. In sum, our main contributions are summarized as follows:

- We treat a graph pooling problem as a multiset encoding problem, under which we consider relationships among nodes in a set with several attention units, to make a compact representation of an entire graph only with one global function, without additional message-passing operations.
- We show that existing GNN with our parametric pooling operation can be as powerful as the WL test, and also be easily extended to the node clustering approaches with learnable clusters.
- We extensively validate GMT for graph classification, reconstruction, and generation tasks on synthetic and real-world graphs, on which it largely outperforms most graph pooling baselines.

3.2 Related Work

Graph Neural Network Existing graph neural network (GNN) models generally encode the nodes by aggregating the features from the neighbors [6, 35, 36, 86], and have achieved a large success on node classification and link prediction tasks. Recently, there also exist transformer-based GNNs [37, 87] that further consider the relatedness between nodes in learning the node embeddings. However, accurately representing the given graph as a whole remains challenging. While using mean or max over the node embeddings allow to represent the entire graph for graph classification [88, 89], they are mostly suboptimal, and may output the same representation for two different graphs. To resolve this problem, recent GNN models [10, 84] aim to make the GNNs to be as powerful as the Weisfeiler-Lehman test [12] in distinguishing graph structures. Yet, they also rely on simple operations, and we need a more sophisticated method to represent the entire graph.

Graph Pooling Graph pooling methods play an essential role of representing the entire graph. While averaging all node features is directly used as simplest pooling methods [81, 90], they result in a loss of information since they consider all node information equally without considering key features for graphs. To overcome this limitation, there have been recent studies on graph pooling to compress the given graph in a task specific manner. Node drop methods use learnable scoring functions to drop nodes with lower scores [79, 85, 82]. Moreover, node clustering methods cast the graph pooling problem into the node clustering problem to map the nodes into a set of clusters [80, 91, 92, 83, 93]. Some methods combine these two approaches by first locally clustering the neighboring nodes, and then dropping unimportant clusters [94]. Meanwhile, edge clustering gradually merges nodes by contracting high-scoring edges between them [95]. In addition, Ahmadi et al. [96] model the memory layer to aggregate nodes without utilizing message-passing after pooling. Finally, there exists a semi-supervised pooling method [97] that scores nodes with an attention scheme [98], to weight more on the important nodes on pooling.

(Multi-)Set Representation Learning Note that a set of nodes in a graph forms a multiset [10]; a set that allows possibly repeating elements. Therefore, contrary to the previous set-encoding methods, which mainly consider non-graph problems [99, 100, 41], we regard the graph representation learning as a multi-set encoding problem. Mathematically, Zaheer et al. [101] and Qi et al. [99] provide the theoretical grounds on permutation invariant functions for the set encoding. Further, Lee et al. [102] propose Set

Transformer, which uses attention mechanism on the set encoding. Building on top of these theoretical grounds on set, we propose the multiset encoding function that explicitly considers the graph structures.

3.3 Graph Multiset Pooling

We posit the graph representation learning problem as a multiset encoding problem, and then utilize the graph-structured attention to consider the global graph structure when encoding the given graph.

3.3.1 Preliminaries

We begin with the general descriptions of graph neural network, and graph pooling.

Graph Neural Network A graph G can be represented by its adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n \times n}$ and the node set $\mathcal{V}$ with $|\mathcal{V}| = n$ nodes, along with the c dimensional node features $\boldsymbol{X} \in \mathbb{R}^{n \times c}$. Graph Neural Networks (GNNs) learn feature representation for different nodes using neighborhood aggregation schemes, which are formalized as the following **Message-Passing** function:

$$\boldsymbol{H}_u^{(l+1)} = \text{UPDATE}^{(l)}\left(\boldsymbol{H}_u^{(l)}, \text{AGGREGATE}^{(l)}\left(\left\{\boldsymbol{H}_v^{(l)}, \forall v \in \mathcal{N}(u)\right\}\right)\right), \tag{3.1}$$

where $\boldsymbol{H}^{(l+1)} \in \mathbb{R}^{n \times d}$ is the node features computed after l-steps of the GNN simplified as follows: $\boldsymbol{H}^{(l+1)} = \text{GNN}^{(l)}(\boldsymbol{H}^{(l)}, \boldsymbol{A}^{(l)})$, UPDATE and AGGREGATE are arbitrary differentiable functions, $\mathcal{N}(u)$ denotes a set of neighboring nodes of u, and $\boldsymbol{H}_u^{(1)}$ is initialized as the input node features $\boldsymbol{X}_u$.

Graph Pooling While message-passing functions can produce a set of node representations, we need an additional READOUT function to obtain an entire graph representation $\boldsymbol{h}_G \in \mathbb{R}^d$ as follows:

$$\boldsymbol{h}_G = \text{READOUT}\left(\{\boldsymbol{H}_v \mid v \in \mathcal{V}\}\right). \tag{3.2}$$

As a READOUT function, we can simply use the average or sum over all node features $\boldsymbol{H}_v, \forall v \in \mathcal{V}$ from the given graph [81, 10]. However, since such aggregation schemes take all node information equally without considering the graph structures, they lose structural information that is necessary for accurately representing a graph. To tackle this limitation, **Node Drop** methods [85, 82] select the high scored nodes $\boldsymbol{i}^{(l+1)} \in \mathbb{R}^{n_{l+1}}$ with learnable score function s at layer l, to drop the unnecessary nodes, denoted as follows:

$$\boldsymbol{y}^{(l)} = s(\boldsymbol{H}^{(l)}, \boldsymbol{A}^{(l)}); \quad \boldsymbol{i}^{(l+1)} = \text{top}_k(\boldsymbol{y}^{(l)}), \tag{3.3}$$

where function s depends on specific implementations, and top_k function samples the top k nodes by dropping nodes with low scores $\boldsymbol{y}^{(l)} \in \mathbb{R}^{n_l}$. Whereas **Node Clustering** methods [80, 83] learn a cluster assignment matrix $\boldsymbol{C}^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ with node features $\boldsymbol{H}^{(l)} \in \mathbb{R}^{n_l \times d}$, to coarsen the nodes and the adjacency matrix $\boldsymbol{A}^{(l)} \in \mathbb{R}^{n_l \times n_l}$ at layer l as follows:

$$\boldsymbol{H}^{(l+1)} = {\boldsymbol{C}^{(l)}}^T \boldsymbol{H}^{(l)}; \quad \boldsymbol{A}^{(l+1)} = {\boldsymbol{C}^{(l)}}^T \boldsymbol{A}^{(l)} \boldsymbol{C}^{(l)}, \tag{3.4}$$

where generating an assignment matrix $\boldsymbol{C}^{(l)}$ depends on specific implementations. While these two approaches obtain decent performances on graph classification tasks, they are suboptimal since node drop methods unnecessarily drop arbitrary nodes, and node clustering methods have limited scalability

to large graphs [103, 82]. Therefore, we need a sophisticated graph pooling layer that coarsens the graph with sparse implementation without discarding nodes.

3.3.2 Graph Multiset Transformer

We now describe the *Graph Multiset Transformer* (GMT) architecture, which can accurately represent the entire graph, given a multiset of node features. We first introduce a multiset encoding scheme that allows to embed two different graphs into distinct embeddings, and then describe the graph multi-head attention that reflects the graph topology in the attention-based multiset encoding.

Multiset Encoding The input of the graph pooling function READOUT consists of nodes in a graph, and they form a multiset (i.e. a set that allows for repeating elements) since different nodes can have identical feature vectors. To design a graph pooling function that is as powerful as the WL test, it needs to satisfy the permutation invariance and injectiveness over the multiset, since two non-isomorphic graphs should be embedded differently through the injective function. While the simple sum pooling satisfies the injectiveness over a multiset [10], it may treat all node embeddings equally without consideration of their relevance to the task. To resolve this issue, we consider attention mechanism on the multiset pooling function to capture structural dependencies among nodes within a graph, in which we can provably enjoy the expressive power of the WL test.

Graph Multi-head Attention To overcome the inability of simple pooling methods (e.g. sum) on distinguishing important nodes, we use the attention mechanism as the main component in our pooling scheme. Assume that we have n node vectors, and the input of the *attention function* (Att) consists of query $\boldsymbol{Q} \in \mathbb{R}^{n_q \times d_k}$, key $\boldsymbol{K} \in \mathbb{R}^{n \times d_k}$ and value $\boldsymbol{V} \in \mathbb{R}^{n \times d_v}$, where n_q is the number of query vectors, n is the number of input nodes, d_k is the dimensionlity of the key vector, and d_v is the dimensionality of the value vector. Then we compute the dot product of the query with all keys, to put more weights on the relevant values, namely nodes, as follows: $\text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = w(\boldsymbol{Q}\boldsymbol{K}^T)\boldsymbol{V}$, where w is an activation function. Instead of computing a single attention, we can further use a multi-head attention [104], by linearly projecting the query $\boldsymbol{Q}$, key $\boldsymbol{K}$, and value $\boldsymbol{V}$ h times respectively to yield h different representation subspaces. The output of the *multi-head attention function* (MH) then can be denoted as follows:

$$\text{MH}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [O_1, ..., O_h]\, \boldsymbol{W}^O; \quad O_i = \text{Att}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V), \tag{3.5}$$

where the operations for h parallel projections are parameter matrices $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_k \times d_k}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d_k \times d_k}$, and $\boldsymbol{W}_i^V \in \mathbb{R}^{d_v \times d_v}$. Also, the output projection matrix is $\boldsymbol{W}^O \in \mathbb{R}^{hd_v \times d_{model}}$, where d_{model} is the output dimensionality for the multi-head attention (MH) function.

While multi-head attention is superior to trivial pooling methods such as sum or mean as it considers global dependencies among nodes, the MH function suboptimally generates the key $\boldsymbol{K}$ and value $\boldsymbol{V}$ for Att, since it linearly projects the obtained node embeddings $\boldsymbol{H}$ from equation 3.1 to further obtain the key and value pairs. To tackle this limitation, we newly define a novel *graph multi-head attention block* (GMH). Formally, given node features $\boldsymbol{H} \in \mathbb{R}^{n \times d}$ with their adjacency information $\boldsymbol{A}$, we construct the key and value using GNNs, to explicitly leverage the graph structure as follows:

$$\text{GMH}(\boldsymbol{Q}, \boldsymbol{H}, \boldsymbol{A}) = [O_1, ..., O_h]\, \boldsymbol{W}^O; \quad O_i = \text{Att}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \text{GNN}_i^K(\boldsymbol{H}, \boldsymbol{A}), \text{GNN}_i^V(\boldsymbol{H}, \boldsymbol{A})), \tag{3.6}$$

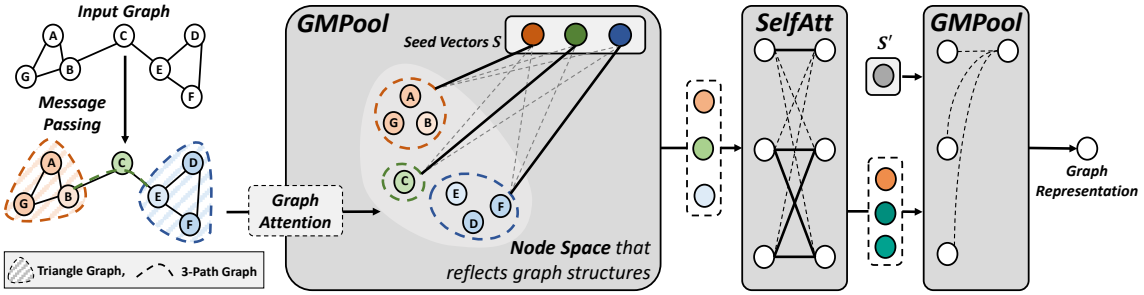

Figure 3.2: **Graph Multiset Transformer.** Given a graph passed through several message passing layers, we use an attention-based pooling block (GMPool) and a self-attention block (SelfAtt) to compress the nodes into few important nodes and consider the interaction among them respectively, within a multiset framework.

where the output of GNN_i contains neighboring information of the graph, compared to the linearly projected node embeddings $\boldsymbol{K}\boldsymbol{W}_i^K$ and $\boldsymbol{V}\boldsymbol{W}_i^V$ in equation 3.5, for key and value matrices in Att.

Graph Multiset Pooling with Graph Multi-head Attention Using the ingredients above, we now propose a graph pooling function that satisfies the injectiveness and permutation invariance, such that the overall architecture can be at most as powerful as the WL test, while taking the graph structure into account. Given node features $\boldsymbol{H} \in \mathbb{R}^{n\times d}$ from GNNs, we define a *Graph Multiset Pooling* (GMPool), which is inspired by the Transformer [104, 102], to compress the n nodes into the k typical nodes, with a parameterized seed matrix $\boldsymbol{S} \in \mathbb{R}^{k\times d}$ for the pooling operation that is directly optimized in an end-to-end fashion, as follows (Figure 3.2-GMPool):

$$\text{GMPool}_k(\boldsymbol{H}, \boldsymbol{A}) = \text{LN}(\boldsymbol{Z} + \text{rFF}(\boldsymbol{Z})); \quad \boldsymbol{Z} = \text{LN}(\boldsymbol{S} + \text{GMH}(\boldsymbol{S}, \boldsymbol{H}, \boldsymbol{A})), \tag{3.7}$$

where rFF is any row-wise feedforward layer that processes each individual row independently and identically, and LN is a layer normalization [105]. Note that the GMH function in equation 3.7 considers interactions between k seed vectors (queries) in $\boldsymbol{S}$ and n nodes (keys) in $\boldsymbol{H}$, to compress n nodes into k clusters with their attention similarities between queries and keys. Also, to extend the pooling scheme from set to multiset, we simply consider redundant node representations.

Self-Attention for Inter-node Relationship While previously described GMPool condenses entire nodes into k representative nodes, a major drawback of this scheme is that it does not consider relationships between nodes. To tackle this limitation, one should further consider the interactions among n or condensed k different nodes. To this end, we propose a *Self-Attention function* (SelfAtt), inspired by the Transformer [104, 102], as follows (Figure 3.2-SelfAtt):

$$\text{SelfAtt}(\boldsymbol{H}) = \text{LN}(\boldsymbol{Z} + \text{rFF}(\boldsymbol{Z})); \quad \boldsymbol{Z} = \text{LN}(\boldsymbol{H} + \text{MH}(\boldsymbol{H}, \boldsymbol{H}, \boldsymbol{H})), \tag{3.8}$$

where, compared to GMH in equation 3.7 that considers interactions between k vectors and n nodes, SelfAtt captures inter-relationships among n nodes by putting node embeddings $\boldsymbol{H}$ on both query and key locations in MH of equation 3.8. To satisfy the injectiveness property of SelfAtt, it might not consider interactions among n nodes, which we discuss in Proposition 3.3.3 of Subsection 3.3.3.

Overall Architecture We now describe the full structure of *Graph Multiset Transformer* (GMT) consisting of GNN and pooling layers using ingredients above (See Figure 3.2). For a graph G with node

features $\boldsymbol{X}$ and an adjacency matrix $\boldsymbol{A}$, the Encoder : $G \mapsto \boldsymbol{H} \in \mathbb{R}^{n \times d}$ is denoted as follows:

$$\text{Encoder}(\boldsymbol{X}, \boldsymbol{A}) = \text{GNN}_2(\text{GNN}_1(\boldsymbol{X}, \boldsymbol{A}), \boldsymbol{A}), \tag{3.9}$$

where we can stack several GNNs to construct the deep structures. After obtaining a set of node features $\boldsymbol{H}$ from an encoder, the pooling layer aggregates the features into a single vector form; Pooling : $\boldsymbol{H}, \boldsymbol{A} \mapsto \boldsymbol{h}_G \in \mathbb{R}^d$. To deal with a large number of nodes, we first condense the entire graph into k representative nodes with *Graph Multiset Pooling* (GMPool), which is also adaptable to the varying size of nodes, and then utilize the interaction among them with *Self-Attention Block* (SelfAtt). Finally, we get the entire graph representation by using GMPool with $k = 1$ as follows:

$$\text{Pooling}(\boldsymbol{H}, \boldsymbol{A}) = \text{GMPool}_1(\text{SelfAtt}(\text{GMPool}_k(\boldsymbol{H}, \boldsymbol{A})), \boldsymbol{A}'), \tag{3.10}$$

where $\boldsymbol{A}' \in \mathbb{R}^{k \times k}$ is the identity or coarsened adjacency matrix since adjacency information should be adjusted after compressing the nodes from n to k with GMPool_k (See Appendix 3.6.1 for details).

3.3.3 Connection with Weisfeiler-Lehman Isomorphism Test

Weisfeiler-Lehman (WL) graph isomorphism test [12] is known for its ability to efficiently distinguish two different graphs. Recent studies [84, 10] show that GNNs can be made to be as powerful as the WL test, by using an injective function over a multiset to map two different graphs into distinct spaces. Building on previous powerful GNNs, if our graph pooling function is injective, then our overall architecture can be at most as powerful as the WL test. To do so, we first recount the theorem from Xu et al. [10], as formalized in Theorem 3.3.1.

Theorem 3.3.1: Non-isomorphic Graphs to Different Embeddings

Let $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ be a GNN, and Weisfeiler-Lehman test decides two graphs $G_1 \in \mathcal{G}$ and $G_2 \in \mathcal{G}$ as non-isomorphic. Then, $\mathcal{A}$ maps two different graphs G_1 and G_2 to distinct vectors if node aggregation and update functions are injective, and graph-level readout, which operates on a multiset of node features $\{\boldsymbol{H}_i\}$, is injective.

Proof. To map two non-isomorphic graphs to distinct embedding spaces with GNNs, we recount the theorem on Graph Isomorphism Network. See Appendix B of Xu et al. [10] for details. □

Since we focus on the representation of graphs through pooling, we deal with the injectiveness of the READOUT function. Our next Lemma 3.3.2 states that GMPool can represent the injective function.

Lemma 3.3.2: Injectiveness on Graph Multiset Pooling

Assume the input feature space $\mathcal{H}$ is a countable set. Then the output of $\text{GMPool}_k^i(\boldsymbol{H}, \boldsymbol{A})$ with $\text{GMH}(\boldsymbol{S}_i, \boldsymbol{H}, \boldsymbol{A})$ for a seed vector $\boldsymbol{S}_i$ can be unique for each multiset $\boldsymbol{H} \subset \mathcal{H}$ of bounded size. Further, the output of full $\text{GMPool}_k(\boldsymbol{H}, \boldsymbol{A})$ constructs a multiset with k elements, which are also unique on the input multiset $\boldsymbol{H}$.

Proof. We first state that the GNNs of the Graph Multi-head Attention (GMH) in a GMPool can represent the injective function over the multiset $\boldsymbol{H}$ with an adjacency information $\boldsymbol{A}$, by selecting proper

message-passing functions that satisfy the WL test [10, 84], denoted as follows: $\boldsymbol{H}' = \text{GNN}(\boldsymbol{H}, \boldsymbol{A})$, where $\boldsymbol{H}' \subset \mathcal{H}$. Then, given enough elements in the multiset, a $\text{GMPool}_k^i(\boldsymbol{H}, \boldsymbol{A})$ can express the sum pooling over the multiset $\boldsymbol{H}'$ defined as follows: $\rho(\sum_{\boldsymbol{h}\in\boldsymbol{H}'} f(\boldsymbol{h}))$, where f and ρ are mapping functions (see the proof of PMA in Lee et al. [102]).

Since $\mathcal{H}$ is a countable set, there is a mapping from elements to prime numbers denoted by $p(\boldsymbol{h}) : \mathcal{H} \to \mathbb{P}$. If we let $f(\boldsymbol{h}) = -\log p(\boldsymbol{h})$, then $\sum_{\boldsymbol{h}\in\boldsymbol{H}'} f(\boldsymbol{h}) = \log \prod_{\boldsymbol{h}\in\boldsymbol{H}'} \frac{1}{p(\boldsymbol{h})}$ which constitutes an unique mapping for every multiset $\boldsymbol{H}' \subset \mathcal{H}$ (see Wagstaff et al. [106]). In other words, $\sum_{\boldsymbol{h}\in\boldsymbol{H}'} f(\boldsymbol{h})$ is injective. Also, we can easily construct a function ρ, such that $\text{GMPool}_k^i(\boldsymbol{H}, \boldsymbol{A}) = \rho(\sum_{\boldsymbol{h}\in\boldsymbol{H}'} f(\boldsymbol{h})) = \rho(\log \prod_{\boldsymbol{h}\in\boldsymbol{H}'} \frac{1}{p(h)})$ is injective for every multiset $\boldsymbol{H} \subset \mathcal{H}$, where $\boldsymbol{H}'$ is derived from the GNN component in the GMPool; $\boldsymbol{H}' = \text{GNN}(\boldsymbol{H}, \boldsymbol{A})$.

Furthermore, since a GMPool considers multiset elements without any order, it satisfies the permutation invariance condition for the multiset function.

Finally, each GMPool block has k components such that the output of it consists of k elements as follows: $\text{GMPool} = \left\{\text{GMPool}_k^i(\boldsymbol{H}, \boldsymbol{A})\right\}_{i=1}^k$, which allows multiple instances for its elements. Then, since each $\text{GMPool}_k^i(\boldsymbol{H}, \boldsymbol{A})$ is unique on the input multiset $\boldsymbol{H}$, the output of the GMPool that consists of k outputs is also unique on the input multiset $\boldsymbol{H}$. □

Thanks to the universal approximation theorem [107], we can construct such functions p and ρ using multi-layer perceptrons (MLPs).

Based upon the injectiveness of GMPool, we further show the injectiveness of SelfAtt, to make an overall architecture (a sequence of GMPool and SelfAtt with GNNs) as powerful as the WL test, formalized in Proposition 3.3.3.

Proposition 3.3.3: Injectiveness on Graph Multiset Transformer

The overall Graph Multiset Transformer with multiple GMPool and SelfAtt can map two different graphs G_1 and G_2 to distinct embedding spaces, such that the resulting GNN with proposed pooling functions can be as powerful as the WL test.

Proof. By Lemma 3.3.2, we know that a *Graph Multiset Pooling* (GMPool) can represent the injective function over the input multiset $\boldsymbol{H} \subset \mathcal{H}$. If we can also show that a *Self-Attention* (SelfAtt) can represent the injective function over the multiset, then the sequence of the GMPool and SelfAtt blocks can satisfy the injectiveness.

Let $\boldsymbol{W}^O$ be a zero matrix in the SelfAtt function. $\text{SelfAtt}(\boldsymbol{H})$ then can be approximated to any instance-wise feed-forward networks denoted as follows: $\text{SelfAtt}(\boldsymbol{H}) = \text{rFF}(\boldsymbol{H})$. Therefore, this rFF is a suitable transformation $\phi : \mathbb{R}^d \to \mathbb{R}^d$ that can be easily constructed over the multiset elements $\boldsymbol{h} \in \boldsymbol{H}$, to satisfy the injectiveness. □

To maximize the discriminative power of the Graph Multiset Transformer (GMT) by satisfying the WL test, we assume that SelfAtt does not consider the interactions among multiset elements, namely nodes. While proper GNNs with the proposed pooling function can be at most as powerful as the WL test with this assumption, our experimental results with the ablation study show that the interaction among nodes is significantly important to distinguish the broad classes of graphs (See Table 3.2).

3.3.4 Connection with Node Clustering Approaches

Node clustering is widely used for coarsening a graph in a hierarchical manner, as described in the equation 3.4. However, since they require to store and even multiply the adjacency matrix $\boldsymbol{A}$ with the soft assignment matrix $\boldsymbol{C}$: $\boldsymbol{A}^{(l+1)} = \boldsymbol{C}^{(l)^T}\boldsymbol{A}^{(l)}\boldsymbol{C}^{(l)}$, they need a quadratic space $\mathcal{O}(n^2)$ for n nodes, which is problematic for large graphs. Meanwhile, our GMPool does not compute a coarsened adjacency matrix $A^{(l+1)}$, such that graph pooling is possible only with a sparse implementation, as formalized in Theorem 3.3.4.

Theorem 3.3.4: Space Complexity of Graph Multiset Pooling

Graph Multiset Pooling condsense a graph with n nodes to k nodes in $\mathcal{O}(nk)$ space complexity, which can be further optimized to $\mathcal{O}(n)$.

Proof. Assume that we have key $\boldsymbol{K} \in \mathbb{R}^{n\times d_k}$ and value $\boldsymbol{V} \in \mathbb{R}^{n\times d_v}$ matrices in the Att function of Graph Multi-head Attention (GMH) for the simplicity, which is described in the equation 3.6. Also, $\boldsymbol{Q}$ is defined as a seed vector $\boldsymbol{S} \in \mathbb{R}^{k\times d}$ in the GMPool function of the equation 3.7. To obtain the weights on the values $\boldsymbol{V}$, we multiply the query $\boldsymbol{Q}$ with key $\boldsymbol{K}$: $\boldsymbol{Q}\boldsymbol{K}^T$. This matrix multiplication then maps a set of n nodes into a set of k nodes, such that it requires $\mathcal{O}(nk)$ space complexity. Also, we can further drop the constant term k: $\mathcal{O}(n)$, by properly setting the small k values; $k \ll n$.

The multiplication of the attention weights $\boldsymbol{Q}\boldsymbol{K}^T$ with value $\boldsymbol{V}$ also takes the same complexity, such that the overall space complexity of GMPool is $\mathcal{O}(nk)$, which can be further optimized to $\mathcal{O}(n)$. □

The space complexity of GNNs with sparse implementation requires $\mathcal{O}(n+m)$ space complexity, where n is the number of nodes, and m is the number of edges in a graph. Therefore, multiple GNNs followed by our GMPool require the total space complexity of $\mathcal{O}(n+m)$ due to the space complexity of the GNN operations. However, GNNs with our GMPool are more efficient than node clustering methods, since node clustering approaches need $\mathcal{O}(n^2)$ space complexity.

In spite of this huge strength on space complexity, our GMPool can be further approximated to the node clustering methods by manipulating an adjacency matrix, as formalized in Proposition 3.3.5.

Proposition 3.3.5: Approximation to Node Clustering

Graph Multiset Pooling GMPool_k can perform hierarchical node clustering with learnable k cluster centroids by Seed Vector S in equation 3.7.

Proof. Node clustering approaches are widely used to coarsen a given large graph in a hierarchical manner with several message-passing functions. The core part of the node clustering schemes is to generate a cluster assignment matrix $\boldsymbol{C}$, to coarsen nodes and adjacency matrix as in an equation 3.4. Therefore, if our Graph Multiset Pooling (GMPool) can generate a cluster assignment matrix $\boldsymbol{C}$, then the proposed GMPool can be directly approximated to the node clustering approaches.

In the proposed GMPool, query $\boldsymbol{Q}$ is generated from a learnable set of k seed vectors $\boldsymbol{S}$, and key $\boldsymbol{K}$ and value $\boldsymbol{V}$ are generated from node features $\boldsymbol{H}$ with GNNs in the Graph Multi-head Attention (GMH) block, as in an equation 3.6. In this function, if we decompose the attention function $\text{Att}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = w(\boldsymbol{Q}\boldsymbol{K}^T)\boldsymbol{V}$ into the dot products of the query with all keys, and the corresponding weighted sum of values, then the first dot product term inherently generates a soft assignment matrix as follows:

$\boldsymbol{C} = w(\boldsymbol{Q}\boldsymbol{K}^T)$. Therefore, the proposed GMPool can be easily extended to the node clustering schemes, with the inherently generated cluster assignment matrix; $\boldsymbol{C} = w(\boldsymbol{Q}\boldsymbol{K}^T)$, where one of the proper choices for the activation function w is the softmax function as follows:

$$w(\boldsymbol{Q}\boldsymbol{K}^T)_{i,j} = \frac{\exp(\boldsymbol{Q}_i \boldsymbol{K}_j^T)}{\sum_{n=1}^{k} \exp(\boldsymbol{Q}_n \boldsymbol{K}_j^T)}. \tag{3.11}$$

Furthermore, through the learnable seed vectors $\boldsymbol{S}$ for the query $\boldsymbol{Q}$, we can learn data dependent k different cluster centroids in an end-to-end fashion. □

Note that, as shown in Subsection 3.4.2 of the main paper, the proposed GMPool significantly outperforms the previous node clustering approaches [80, 83]. This is because, contrary to them, the proposed GMPool can explicitly learn data dependent k cluster centroids by learnable seed vectors $\boldsymbol{S}$.

3.4 Experiment

To validate the proposed *Graph Multiset Transformer* (GMT) for graph representation learning, we evaluate it on classification, reconstruction and generation tasks of synthetic and real-world graphs.

3.4.1 Graph Classification

Objective The goal of graph classification is to predict a label $\boldsymbol{y}_i \in \mathcal{Y}$ of a given graph $G_i \in \mathcal{G}$, with a mapping function $f : \mathcal{G} \rightarrow \mathcal{Y}$. To this end, we use a set of node representations $\{\boldsymbol{H}_v \mid v \in \mathcal{V}\}$ to obtain an entire graph representation $\boldsymbol{h}_G$ that is used to classify a label $f(G) = \hat{\boldsymbol{y}}$. We then learn f with a cross-entropy loss, to minimize the negative log likelihood as follows: $\min \sum_{i=1} -\boldsymbol{y}_i \log \hat{\boldsymbol{y}}_i$.

Datasets Among TU datasets [108], we select 6 datasets including 3 datasets (D&D, PROTEINS, and MUTAG) on Biochemical domain, and 3 datasets (IMDB-B, IMDB-M, and COLLAB) on Social domain with accuracy for evaluation metric. Also, we use 4 molecule datasets (HIV, Tox21, ToxCast, BBBP) from the OGB datasets [109] with ROC-AUC for evaluation metric. Statistics are reported in the Table 3.1, and more details are described in the Appendix 3.6.3.

Models **1) GCN. 2) GIN.** GNNs with mean or sum pooling [6, 10]. **3) Set2Set.** Set pooling baseline [110]. **4) SortPool. 5) SAGPool. 6) TopKPool. 7) ASAP.** The methods [79, 82, 85, 94] that use the node drop, by dropping nodes (or clusters) with lower scores using scoring functions. **8) DiffPool. 9) MinCutPool. 10) HaarPool. 11) StructPool.** The methods [80, 83, 92, 93] that use the node clustering, by grouping a set of nodes into a set of clusters using a cluster assignment matrix. **12) EdgePool.** The method [95] that gradually merges two adjacent nodes that have a high score edge. **13) GMT.** The proposed Graph Multiset Transformer (See Appendix 3.6.2 for detailed descriptions).

Implementation Details For a fair comparison of pooling baselines [82], we fix the GCN [6] as a message passing layer. We evaluate the model performance on TU datasets for 10-fold cross validation [79, 10] with LIBSVM [111]. Also, we use the initial node features following the fair comparison setup [112]. We evaluate the performance on OGB datasets with their original feature extraction and data split settings [109]. Experimental details are described in the Appendix 3.6.3.

Table 3.1: **Graph classification results** on test sets. The reported results are mean and standard deviation over 10 different runs. Best performance and its comparable results ($p > 0.05$) from the t-test are marked in bold. Hyphen (-) denotes out-of-resources that take more than 10 days (See Figure 3.4 for the time efficiency analysis).

	Biochemical Domain							Social Domain			Significance
	D&D	**PROTEINS**	**MUTAG**	**HIV**	**Tox21**	**ToxCast**	**BBBP**	**IMDB-B**	**IMDB-M**	**COLLAB**	
# graphs	1,178	1,113	188	41,127	7,831	8,576	2,039	1,000	1,500	5,000	-
# classes	2	2	2	2	12	617	2	2	3	3	-
Avg # nodes	284.32	39.06	17.93	25.51	18.57	18.78	24.06	19.77	13.00	74.49	-
GCN	72.05 ± 0.55	73.24 ± 0.73	69.50 ± 1.78	**76.81** ± 1.01	75.04 ± 0.80	60.63 ± 0.51	65.47 ± 1.73	**73.26** ± 0.46	50.39 ± 0.41	**80.59** ± 0.27	3 / 10
GIN	70.79 ± 1.17	71.46 ± 1.66	81.39 ± 1.53	75.95 ± 1.35	73.27 ± 0.84	60.83 ± 0.46	**67.65** ± 3.00	**72.78** ± 0.86	48.13 ± 1.36	78.19 ± 0.63	2 / 10
Set2Set	71.94 ± 0.56	73.27 ± 0.85	69.89 ± 1.94	74.70 ± 1.65	74.10 ± 1.13	59.70 ± 1.04	66.79 ± 1.05	**72.90** ± 0.75	50.19 ± 0.39	79.55 ± 0.39	1 / 10
SortPool	75.58 ± 0.72	73.17 ± 0.88	71.94 ± 3.55	71.82 ± 1.63	69.54 ± 0.75	58.69 ± 1.71	65.98 ± 1.70	72.12 ± 1.12	48.18 ± 0.83	77.87 ± 0.47	0 / 10
DiffPool	77.56 ± 0.41	73.03 ± 1.00	79.22 ± 1.02	75.64 ± 1.86	74.88 ± 0.81	62.28 ± 0.56	**68.25** ± 0.96	**73.14** ± 0.70	**51.31** ± 0.72	78.68 ± 0.43	3 / 10
SAGPool(G)	71.54 ± 0.91	72.02 ± 1.08	76.78 ± 2.12	74.56 ± 1.69	71.10 ± 1.06	59.88 ± 0.79	65.16 ± 1.93	72.16 ± 0.88	49.47 ± 0.56	78.85 ± 0.56	0 / 10
SAGPool(H)	74.72 ± 0.82	71.56 ± 1.49	73.67 ± 4.28	71.44 ± 1.67	69.81 ± 1.75	58.91 ± 0.80	63.94 ± 2.59	**72.55** ± 1.28	50.23 ± 0.44	78.03 ± 0.31	1 / 10
TopKPool	73.63 ± 0.55	70.48 ± 1.01	67.61 ± 3.36	72.27 ± 0.91	69.39 ± 2.02	58.42 ± 0.91	65.19 ± 2.30	71.58 ± 0.95	48.59 ± 0.72	77.58 ± 0.85	0 / 10
MinCutPool	**78.22** ± 0.54	**74.72** ± 0.48	79.17 ± 1.64	75.37 ± 2.05	75.11 ± 0.69	62.48 ± 1.33	65.97 ± 1.13	72.65 ± 0.75	**51.04** ± 0.70	**80.87** ± 0.34	4 / 10
StructPool	**78.45** ± 0.40	**75.16** ± 0.86	79.50 ± 1.75	75.85 ± 1.81	75.43 ± 0.79	62.17 ± 1.61	**67.01** ± 2.65	72.06 ± 0.64	50.23 ± 0.53	77.27 ± 0.51	3 / 10
ASAP	76.58 ± 1.04	73.92 ± 0.63	77.83± 1.49	72.86 ± 1.40	72.24 ± 1.66	58.09 ± 1.62	63.50 ± 2.47	72.81 ± 0.50	**50.78** ± 0.75	78.64 ± 0.50	1 / 10
EdgePool	75.85 ± 0.58	**75.12** ± 0.76	74.17± 1.82	72.66 ± 1.70	73.77 ± 0.68	60.70 ± 0.92	**67.18** ± 1.97	72.46 ± 0.74	**50.79** ± 0.59	-	3 / 9
HaarPool	-	-	66.11± 1.50	-	-	-	66.11 ± 0.82	**73.29** ± 0.34	49.98 ± 0.57	-	1 / 5
GMT (Ours)	**78.72** ± 0.59	**75.09** ± 0.59	**83.44** ± 1.33	**77.56** ± 1.25	**77.30** ± 0.59	**65.44** ± 0.58	**68.31** ± 1.62	**73.48** ± 0.76	**50.66** ± 0.82	**80.74** ± 0.54	**10 / 10**

Model	**D&D**	**PROTEINS**	**BBBP**
GMT	**78.72**	**75.09**	**68.31**
w/o message passing	78.06	75.07	65.26
w/o graph attention	78.08	74.50	66.21
w/o self-attention	75.13	74.22	64.53
mean pooling	72.05	73.24	65.47

Table 3.2: Ablation Study of GMT on the D&D, PROTEINS, and BBBP datasets for graph classification.

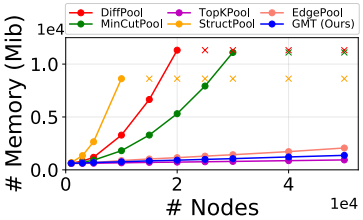

Figure 3.3: Memory efficiency of GMT compared with baselines. X indicates out-of-memory error.

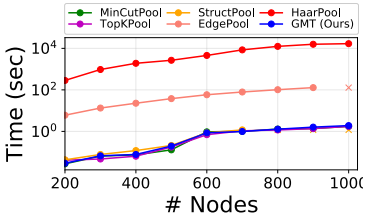

Figure 3.4: Time efficiency of GMT compared with baselines. X indicates out-of-memory error.

Classification Results Table 3.1 shows that our GMT outperforms most baselines, or achieves comparable performance to the best baseline results. These results demonstrate that our method is simple yet powerful as it only performs a single global operation at the final layer, unlike several baselines that use multiple pooling with a sequence of message passing (See Figure 3.9 for the detailed model architectures). Note that, since graph classification tasks mostly require the discriminative information to predict the labels of a graph, GNN baselines without parametric pooling, such as GCN and GIN, sometimes outperform pooling baselines on some datasets. In addition, recent work [113], which reveals that message-passing layers are dominant in the graph classification, supports this phenomenon. Therefore, we conduct experiments on graph reconstruction to directly quantify the amount of retained information after pooling, which we describe in the next subsection.

Ablation Study To see where the performance improvement comes from, we conduct an ablation study on GMT by removing graph attention, self attention, and message-passing operations. Table 3.2 shows that using graph attention with self-attention helps significantly improve the performances from the mean pooling. Further, performances of the GMT without message-passing layers indicate that our pooling layer well captures the graph multiset structure only with pooling without GNNs.

Efficiency While node clustering methods achieve decent performances in Table 3.1, they are known to suffer from large memory usage since they cannot work with sparse graph implementations. To compare the GPU **Memory Efficiency** of GMT with baseline models, we test it on the Erdos-Renyi graphs [114] (See Appendix 3.6.3 for detail setup). Figure 3.3 shows that our GMT is highly efficient in terms of memory thanks to its compatibility with sparse graphs, making it more practical over memory-

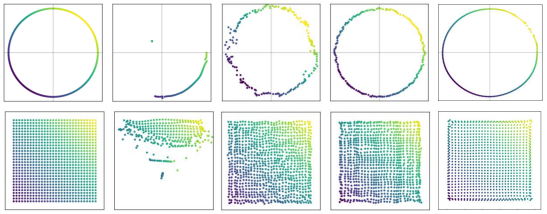

Figure 3.5: Reconstruction results of ring and grid synthetic graphs, compared to node drop and clustering methods. See Figure 3.10 for high resolution.

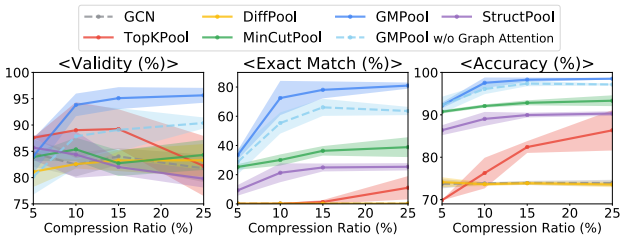

Figure 3.6: Reconstruction results on the ZINC molecule dataset by varying the compression ratio. Solid lines denote the mean, and shaded areas denote the variance.

heavy pooling baselines. In addition to this, we measure the **Time Efficiency** to further validate the practicality of GMT in terms of time complexity. We validate it with the same Erdos-Renyi graphs (See Appendix 3.6.3 for detail setup). Figure 3.4 shows that GMT takes less than (or nearly about) a second even for large graphs, compared to the slowly working models such as HaarPool and EdgePool. This result further confirms that our GMT is practically efficient.

3.4.2 Graph Reconstruction

Graph classification does not directly measure the expressiveness of GNNs since identifying discriminative features may be more important than accurately representing graphs. Meanwhile, graph reconstruction directly quantifies the graph information retained by condensed nodes after pooling.

Objective For graph reconstruction, we train an autoencoder to reconstruct the input node features $\boldsymbol{X} \in \mathbb{R}^{n \times c}$ from their pooled representations $\boldsymbol{X}^{pool} \in \mathbb{R}^{k \times c}$. The learning objective to minimize the discrepancy between the original graph $\boldsymbol{X}$ and the reconstructed graph $\boldsymbol{X}^{rec}$ with a cluster assignment matrix $\boldsymbol{C} \in \mathbb{R}^{n \times k}$ is denoted as follows: $\min \|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$, where $\boldsymbol{X}^{rec} = \boldsymbol{C}\boldsymbol{X}^{pool}$.

Experimental Setup We first experiment with **Synthetic Graph**, such as ring and grid [83], that can be represented in a 2-D Euclidean space, where the goal is to restore the location of each node from pooled features, with an adjacency matrix. We further experiment with real-world **Molecule Graph**, namely ZINC datasets [115], which consists of 12K molecular graphs. See Appendix 3.6.4 for the experimental details including model descriptions.

Reconstruction Results Figure 3.5 shows the original and the reconstructed graphs for **Synthetic Graph** of ring and grid structures. The noisy results of baselines indicate that the condensed node features do not fully capture the original graph structure. Whereas our GMPool yields almost perfect reconstruction, which demonstrates that our pooling operation learns meaningful representation without discarding the original graph information. We further validate the reconstruction performance of the proposed GMPool on the real-world **Molecule Graph**, namely ZINC, by varying the compression ratio. Figure 3.6 shows reconstruction results on the molecule graph, on which GMPool largely outperforms all compared baselines in terms of validity, exact match, and accuracy (High score indicates the better, and see Appendix 3.6.4 for the detailed description of evaluation metrics). With given results, we demonstrate that our GMPool can be easily extended to the node clustering schemes, while it is powerful enough to encode meaningful information to reconstruct the graph.

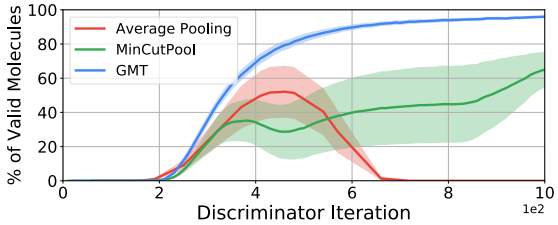

Figure 3.8: Validity curve for molecule generation on QM9 dataset from MolGAN. Solid lines denote the mean and shaded areas denote the variance.

Top-k accuracy:		1	3	5	10	20	50
Reaction Class Unknown	GLN	51.41	67.55	74.92	83.48	88.64	92.37
	MinCutPool	51.17	67.47	**75.59**	**83.68**	89.31	92.31
	GMT (Ours)	**51.83**	**68.20**	75.17	83.20	**89.33**	**92.47**
Reaction Class as Prior	GLN	63.53	78.27	84.32	89.51	92.17	93.17
	MinCutPool	63.91	79.19	84.76	89.69	92.13	93.23
	GMT (Ours)	**64.17**	**79.61**	**85.32**	**89.97**	**92.31**	**93.25**

Table 3.3: Top-k accuracy for retrosynthesis experiment on USPTO-50k data, for cases where the reaction class is given as prior information (Bottom) and not given (Top).

Qualitative Analysis We visualize the reconstruction examples from ZINC in Figure 3.7, where colors in the left figure indicate the assigned clusters on each atoms, and red dashed circles indicate the incorrectly predicted atoms on the reconstructed molecule. As shown in Figure 3.7, GMPool yields more calibrated clustering than MinCutPool, capturing the detailed substructures, which results in the successful reconstruction (See Figure 3.11 in Appendix D for more reconstruction examples).

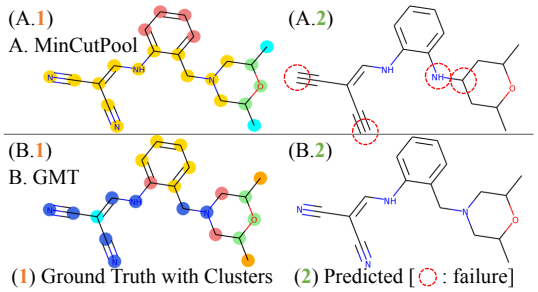

Figure 3.7: Reconstruction example with assigned clusters as colors on left and reconstructed molecules on right.

3.4.3 Graph Generation

Objective Graph generation is used to generate a valid graph that satisfies the desired properties, in which graph encoding is used to improve the generation performances. Formally, given a graph G with graph encoding function f, the goal here is to generate a valid graph $\bar{G} \in \mathcal{G}$ of desired property $\boldsymbol{y}$ with graph decoding function g as follows: $\min d(\boldsymbol{y}, \bar{G})$, where $\bar{G} = g(f(G))$. d is a distance metric between the generated graph and desired properties, to guarantee that the graph has them.

Experimental Setup To evaluate the applicability of our model, we experiment on **Molecule Generation** to stably generate the valid molecules with MolGAN [116], and **Retrosynthesis** to empower the synthesis performances with Graph Logic Network (GLN) [117], by replacing their graph embedding function $f(G)$ to ours. In both experiments, we replace the average pooling to either the MinCutPool or GMT. See Appendix 3.6.5 for more experimental details.

Generation Results The power of a discriminator distinguishing whether a molecule is real or fake is highly important to create a valid molecule in MolGAN. Figure 3.8 shows the validity curve on the early stage of MolGAN training for **Molecule Generation**, and the representation power of GMT significantly leads to the stabilized generation of valid molecules than baselines. Further, Table 3.3 shows **Retrosynthesis** results, where we use the GLN as a backbone architecture. Similar to the molecule generation, retrosynthesis with GMT further improves the performances, which suggests that GMT can replace existing pooling methods for improved performances on diverse graph tasks.

3.5 Conclusion

In this work, we pointed out that existing graph pooling approaches either do not consider the task relevance of each node (sum or mean) or may not satisfy the injectiveness (node drop and clustering methods). To overcome such limitations, we proposed a novel graph pooling method, *Graph Multiset Transformer* (GMT), which not only encodes the given set of node embeddings as a multiset to uniquely embed two different graphs into two distinct embeddings, but also considers both the global structure of the graph and their task relevance in compressing the node features. We theoretically justified that the proposed pooling function is as powerful as the WL test, and can be extended to the node clustering schemes. We validated the proposed GMT on 10 graph classification datasets, and our method outperformed state-of-the-art graph pooling models on most of them. We further showed that our method is superior to the existing graph pooling approaches on graph reconstruction and generation tasks, which require more accurate representations of the graph than classification tasks. We strongly believe that the proposed pooling method will bring substantial practical impact, as it is generally applicable to many graph-learning tasks that are becoming increasingly important.

3.6 Appendix

3.6.1 Details for Graph Multiset Transformer Components

In this section, we describe the *Graph Multiset Pooling* (GMPool) and *Self-Attention* (SelfAtt), which are the main components of the proposed *Graph Multiset Transformer*, in detail.

Graph Multiset Pooling The core components of the Graph Multiset Pooling (GMPool) is the Graph Multi-head Attention (GMH) that considers the graph structure into account, by constructing the key $\boldsymbol{K}$ and value $\boldsymbol{V}$ using GNNs, as described in the equation 3.6. As shown in the Table 3.2 of the main paper, this graph multi-head attention significantly outperforms the naive multi-head attention (MH in equation 3.5). However, after compressing the n nodes into the k nodes with GMPool_k, we can not directly perform further GNNs since the original adjacency information is useless after pooling. To tackle this limitation, we can generate the new adjacency matrix $\boldsymbol{A}'$ for the compressed nodes, by performing node clustering as described in Proposition 5 of the main paper as follows:

$$\text{GMPool}_1(\text{GMPool}_k(\boldsymbol{H}, \boldsymbol{A}), \boldsymbol{A}'); \quad \boldsymbol{A}' = \boldsymbol{C}^T \boldsymbol{A} \boldsymbol{C}, \tag{3.12}$$

where $\boldsymbol{C}$ is the generated cluster assignment matrix, and $\boldsymbol{A}'$ is the coarsened adjacency matrix as described in the equation 3.4. However, this approach is well known for their scalability issues [82, 103], since they require quadratic space $\mathcal{O}(n^2)$ to store and even multiply the adjacency matrix $\boldsymbol{A}$ with the soft assignment matrix $\boldsymbol{C}$. Therefore, we leave doing this as a future work, and use the following trick. By replacing the adjacency matrix $\boldsymbol{A}$ with the identity matrix $\boldsymbol{I}$ in the GMPool except for the first block, we can easily perform multiple GMPools without any constraints, which is approximated to the GMPool with MH in the equation 3.5, rather than GMH in the equation 3.6, as follows:

$$\text{GMPool}_1(\text{GMPool}_k(\boldsymbol{H}, \boldsymbol{A}), \boldsymbol{A}'); \quad \boldsymbol{A}' = \boldsymbol{I}. \tag{3.13}$$

Self-Attention The Self-Attention (SelfAtt) function can consider the inter-relationships between nodes in a set, which helps the network to take the global graph structure into account. Because of this advantage, the self-attention function significantly improves the proposed model performance on the graph classification tasks, as shown in the Table 3.2 of the main paper. From a different perspective, we can regard the Self-Attention function as a graph neural network (GNN) with a complete graph. Specifically, given k nodes from the previous layer, the Multi-head Attention (MH) of the Self-Attention function first constructs the adjacency matrix among all nodes with their similarities, through the matrix multiplication of the query with key: $\boldsymbol{Q}\boldsymbol{K}^T$, and then computes the outputs with the sum of the obtained weights on the value. In other words, the self-attention function can be considered as one message passing function with a soft adjacency matrix, which might be further connected to the Graph Attention Network [36].

3.6.2 Baselines and Our Model

1) GCN. This method [6] is the mean pooling baseline with Graph Convolutional Network (GCN) as a message passing layer.

2) GIN. This method [10] is the sum pooling baseline with Graph Isomorphism Network (GIN) as a message passing layer.

3) Set2Set. This method [110] is the set pooling baseline that uses a recurrent neural network to encode a set of all nodes, with content-based attention over them.

4) SortPool. This method [79] is the node drop baseline that drops unimportant nodes by sorting their representations, which are directly generated from the previous GNN layers.

5) SAGPool. This method [82] is the node drop baseline that selects the important nodes, by dropping unimportant nodes with lower scores that are generated by the another graph convolutional layer, instead of using scores from the previously passed layers. Particularly, this method has two variants. **5.1) SAGPool(G)** is the global node drop method that drops unimportant nodes one time at the end of their architecture. **5.2) SAGPool(H)** is the hierarchical node drop method that drops unimportant nodes sequentially with multiple graph convolutional layers.

6) TopkPool. This method [85] is the node drop baseline that selects the top-ranked nodes using a learnable scoring function.

7) ASAP. This method [94] is the node drop baseline that first locally generates the clusters with neighboring nodes, and then drops the lower score clusters using a scoring function.

8) DiffPool. This method [80] is the node clustering baseline that produces the hierarchical representation of the graphs in an end-to-end fashion, by clustering similar nodes into the few nodes through graph convolutional layers.

9) MinCutPool. This method [83] is the node clustering baseline that applies the spectral clustering with GNNs, to coarsen the nodes and the adjacency matrix of a graph.

10) HaarPool. This method [92] is the spectral-based pooling baseline that compresses the node features with a nonlinear transformation in a Haar wavelet domain. Since it directly uses the spectral clustering to generate a coarsened matrix, the time complexity cost is relatively higher than other pooling methods.

11) StructPool. This method [93] is the node clustering baseline that integrates the concept of the conditional random field into the graph pooling. While this method can be used with a hierarchical scheme, we use it with a global scheme following their original implementation, which is similar to the SortPool [79].

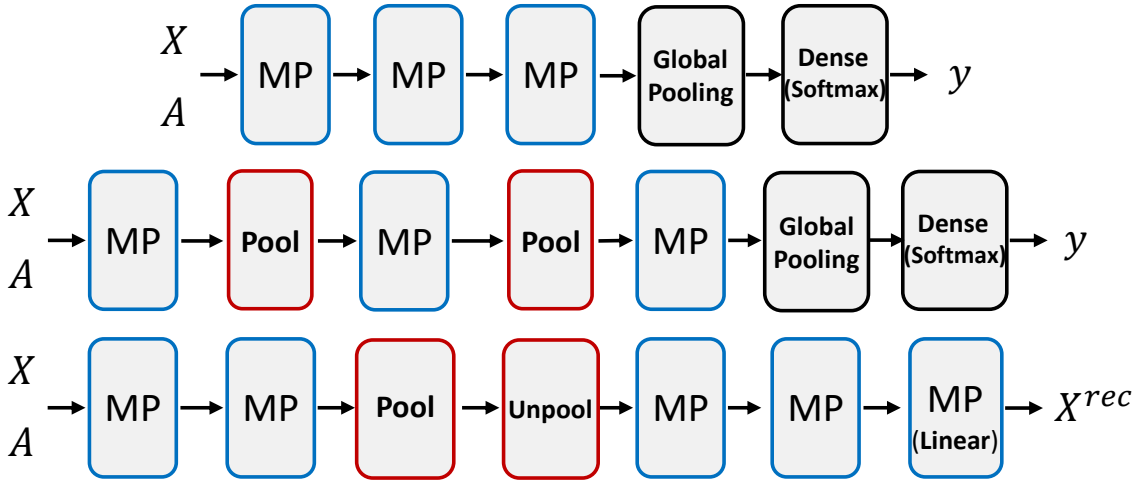

Figure 3.9: **Illustration of High-level Model Architectures. (Top):** Global Graph Classification; GCN, GIN, Set2Set, SortPool, SAGPool(G), StructPool, GMT. **(Middle:)** Hierarchical Graph Classification; DiffPool, SAGPool(H), TopKPool, MinCutPool, ASAP, EdgePool, HaarPool. **(Bottom:)** Graph Reconstruction; DiffPool, TopKPool, MinCutPool, GMT. MP denotes the message passing layer.

12) EdgePool. This method [95] is the edge clustering baseline that gradually merges the nodes, by contracting the high score edge between two adjacent nodes.

13) GMT. Our Graph Multiset Transformer that first condenses all nodes into the important nodes by GMPool, and then considers interactions between nodes in a set. Since it operates on the global READOUT layer, it can be coupled with hierarchical pooling methods by replacing their last layer.

3.6.3 Experimental Details on Graph Classification

Dataset Among TU datasets [108], we select the 6 datasets including 3 datasets (D&D, PROTEINS, and MUTAG) on Biochemical domain, and 3 datasets (IMDB-B, IMDB-M, and COLLAB) on Social domain. We use the classification accuracy as an evaluation metric. As suggested by Errica et al. [112] for a fair comparison, we use the one-hot encoding of their atom types as initial node features in the bio-chemical datasets, and the one-hot encoding of node degrees as initial node features in the social datasets. Moreover, we use the recently suggested 4 molecule graphs (HIV, Tox21, ToxCast, BBBP) from the OGB datasets [109]. We use the ROC-AUC for an evaluation metric, and use the additional atom and bond features, as suggested by Hu et al. [109]. Dataset statistics are reported in the Table 3.1 of the main paper.

Implementation Details on Classification Experiments For all experiments on TU datasets, we evaluate the model performance with a 10-fold cross validation setting, where the dataset split is based on the conventionally used training/test splits [118, 79, 10], with LIBSVM [111]. In addition, we use the 10 percent of the training data as a validation data following the fair comparison setup [112]. For all experiments on OGB datasets, we evaluate the model performance following the original training/validation/test dataset splits [109]. We use the early stopping criterion, where we stop the training if there is no further improvement on the validation loss during 50 epochs, for the TU datasets. Further, the maximum number of epochs is set to 500. We then report the average performances on the validation and test sets, by performing overall experiments 10 times with different seeds.

For all experiments on TU datasets except the D&D, the learning rate is set to 5×10^{-4}, hidden

size is set to 128, batch size is set to 128, weight decay is set to 1×10^{-4}, and dropout rate is set to 0.5. Since the D&D dataset has a large number of nodes (See Table 3.1 in the main paper), node clustering methods can not perform clustering operations on large graphs with large batch sizes, such that the hidden size is set to 32, and batch size is set to 10 on the D&D dataset. For all experiments on OGB datasets except the HIV, the learning rate is set to 1×10^{-3}, hidden size is set to 128, batch size is set to 128, weight decay is set to 1×10^{-4}, and dropout rate is set to 0.5. Since the HIV dataset contains a large number of graphs compared to others (See Table 3.1 in the main paper), the batch size is set to 512 for fast training. Then we optimize the network with Adam optimizer [75]. For a fair comparison of baselines [82], we use the three GCN layers [6] as a message passing function for all models with jumping knowledge strategies [119], and only change the pooling architecture throughout all models, as illustrated in Figure 3.9. Also, we set the pooling ratio as 25% in each pooling layer for both baselines and our models.

Implementation Details on Efficiency Experiments To compare the GPU **memory efficiency** of GMT against baseline models including node drop and node clustering methods, we first generate the Erdos-Renyi graphs [114] by varying the number of nodes n, where the edge size m is twice the number of nodes: $m = 2n$. For all models, we compress the given n nodes into the $k = 4$ nodes at the first pooling function.

To compare the **time efficiency** of GMT against baseline models, we first generate the Erdos-Renyi graphs [114] by varying the number of nodes n with $m = n^2/10$ edges, following the setting of HaarPool [92]. For all models, we set the pooling ratio as 25% except for HaarPool, since it compresses the nodes according to the coarse-grained chain of a graph. We measure the forward time, including CPU and GPU, for all models with 50 graphs over one batch.

3.6.4 Experimental Details on Graph Reconstruction

Dataset We first experiment with synthetic graphs represented in a 2-D Euclidean space, such as ring and grid structures. The node features of a graph consist of their location in a 2-D coordinate space, and the adjacency matrix indicates the connectivity pattern of nodes. The goal here is to restore all node locations from compressed features after pooling, with the intact adjacency matrix.

While synthetic graphs are appropriate choices for the qualitative analysis, we further do the quantitative evaluation of models with real-world molecular graphs. Specifically, we use the subset [120] of the ZINC dataset [115], which consists of 12K real-world molecular graphs, to further conduct a graph reconstruction on the large number of various graphs. The goal of the molecule reconstruction task is to restore the exact atom types of all nodes in the given graph, from the compressed representations after pooling.

Common Implementation Details Following Bianchi et al. [83], we use the two message passing layers both right before the pooling operation and right after the unpooling operation. Also, both pooling and unpooling operations are performed once and sequentially connected, as illustrated in the Figure 3.9. We compare our methods against both the node drop (TopKPool [85]) and node clustering (DiffPool [80] and MinCutPool [83]) methods. For the node drop method, we use the unpooling operation proposed in the graph U-net [85]. For the node clustering methods, we use the graph coarsening schemes described in the equation 3.4, with their specific implementations on generating an assignment matrix. For our proposed method, we only use the one Graph Multiset Pooling (GMPool) without SelfAtt, where we

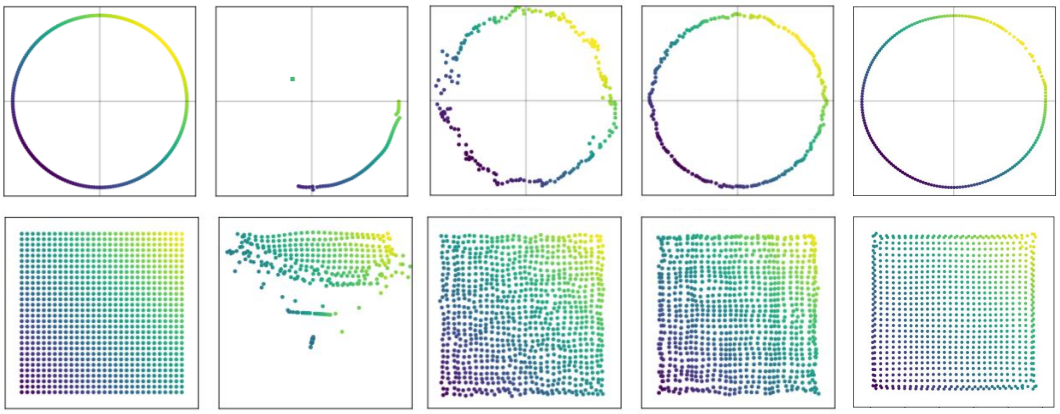

Figure 3.10: High resolution images for synthetic graph reconstruction results in Figure 3.5.

follow the node clustering approaches as described in the subsection 3.3.4 by generating a single soft assignment matrix with one head $h = 1$ in the multi-head attention function. For experiments of both synthetic and molecule reconstructions, the learning rate is set to 5×10^{-3}, and hidden size is set to 32. We then optimize the network with Adam optimizer [75].

Implementation Details on Synthetic Graph We set the pooling ratio of all models as 25%. For the loss function, we use the Mean Squared Error (MSE) to train models. We use the early stopping criterion, where we stop the training if there is no further improvement on the training loss during 1,000 epochs. Further, the maximum number of epochs is set to 10,000. Note that, there is no other available graphs for validation of the synthetic graph, such that we train and test the models only with the given graph in the Figure 3.10. The baseline results are adopted from Bianchi et al. [83].

Implementation Details on Molecule Graph We set the pooling ratio of all models as 5%, 10%, 15%, and 25%, and plot all results in the Figure 3.6 of the main paper. Note that, in the case of molecule graph reconstruction, a softmax layer is appended at the last layer of the model architecture to classify the original atom types of all nodes. For the loss function, we use the cross entropy loss to train models. We use the early stopping criterion, where we stop the training if there is no further improvement on the validation loss during 50 epochs. Further, the maximum number of epochs is set to 500, and batch size is set to 128. Note that, in the case of molecule graph reconstruction on the ZINC dataset, we strictly separate the training, validation and test sets, as suggested by Dwivedi et al. [120]. We perform all experiments 5 times with 5 different random seeds, and then report the averaged result with the standard deviation. Note that, in addition to baselines mentioned in the common implementation details paragraph, we compare two more baselines: GCN with a random assignment matrix for pooling, which is adopted from Mesquita et al. [113], and StructPool [93], for the real-world molecule graph reconstruction.

Evaluation Metrics for Molecule Reconstruction For quantitative evaluations, we use the three metrics as follows: *1) validity* indicates the number of reconstructed molecules that are chemically valid, *2) exact match* indicates the number of reconstructed molecules that are exactly same as the original molecules, and *3) accuracy* indicates the classification accuracy of atom types of all nodes.

Table 3.4: Graph classification results on validation sets with standard deviations. All results are averaged over 10 different runs. Best performance and its comparable results ($p > 0.05$) from the t-test are marked in blod. Hyphen (-) denotes out-of-resources that take more than 10 days (See Figure 3.4 for the time efficiency analysis).

	Biochemical Domain							Social Domain		
	D&D	**PROTEINS**	**MUTAG**	**HIV**	**Tox21**	**ToxCast**	**BBBP**	**IMDB-B**	**IMDB-M**	**COLLAB**
# graphs	1,178	1,113	188	41,127	7,831	8,576	2,039	1,000	1,500	5,000
# classes	2	2	2	2	12	617	2	2	3	3
Avg # nodes	284.32	39.06	17.93	25.51	18.57	18.78	24.06	19.77	13.00	74.49
GCN	76.17 ± 0.65	77.13 ± 0.44	76.56 ± 1.75	81.27 ± 0.92	78.80 ± 0.40	65.66 ± 0.40	93.35 ± 1.08	77.93 ± 0.28	54.29 ± 0.23	83.08 ± 0.13
GIN	76.85 ± 0.61	78.43 ± 0.45	**94.44** ± 0.52	82.10 ± 1.01	78.20 ± 0.45	66.29 ± 0.42	94.64 ± 0.36	**78.38** ± 0.26	54.04 ± 0.29	82.19 ± 0.25
Set2Set	76.32 ± 0.40	77.64 ± 0.41	79.72 ± 2.40	80.07 ± 0.93	79.13 ± 0.75	66.39 ± 0.49	91.89 ± 1.48	78.13 ± 0.30	54.39 ± 0.19	82.34 ± 0.23
SortPool	80.68 ± 0.59	77.92 ± 0.42	81.33 ± 3.00	81.17 ± 2.30	75.97 ± 0.76	64.26 ± 1.17	94.21 ± 1.04	77.46 ± 0.60	52.95 ± 0.62	80.58 ± 0.25
DiffPool	81.33 ± 0.33	79.09 ± 0.36	87.94 ± 1.93	**83.16** ± 0.44	80.02 ± 0.38	**69.73** ± 0.79	**96.32** ± 0.36	77.86 ± 0.39	54.77 ± 0.19	81.69 ± 0.31
SAGPool(G)	76.73 ± 0.80	77.01 ± 0.58	88.11 ± 1.21	80.55 ± 1.89	77.03 ± 0.76	65.51 ± 0.91	**95.59** ± 1.22	**78.09** ± 0.58	53.73 ± 0.42	81.91 ± 0.45
SAGPool(H)	79.56 ± 0.67	77.24 ± 0.56	86.06 ± 2.07	79.21 ± 1.50	75.36 ± 2.63	64.05 ± 0.83	93.05 ± 3.00	77.11 ± 0.46	53.49 ± 0.65	80.55 ± 0.56
TopKPool	78.54 ± 0.73	75.47 ± 0.90	75.06 ± 2.12	79.24 ± 1.84	75.06 ± 2.30	64.56 ± 0.56	93.31 ± 2.32	76.12 ± 0.79	52.75 ± 0.58	79.94 ± 0.86
MinCutPool	81.96 ± 0.39	79.23 ± 0.66	87.22 ± 1.72	**83.12** ± 1.27	**81.10** ± 0.42	**69.09** ± 1.12	**95.99** ± 0.47	77.76 ± 0.36	**54.94** ± 0.19	**83.37** ± 0.18
StructPool	**82.56** ± 0.37	**80.00** ± 0.27	91.50 ± 0.95	81.09 ± 1.26	79.61 ± 0.70	66.49 ± 1.59	95.18 ± 0.59	77.14 ± 0.31	54.13 ± 0.39	79.90 ± 0.18
ASAP	81.58 ± 0.38	78.71 ± 0.45	91.33± 0.65	79.80 ± 1.88	77.33 ± 1.34	63.82 ± 0.75	92.96 ± 1.09	77.89 ± 0.51	**55.17** ± 0.33	82.11 ± 0.33
EdgePool	80.32 ± 0.44	79.61 ± 0.25	87.28± 1.18	81.84 ± 1.32	78.92 ± 0.29	66.21 ± 0.64	94.98 ± 0.62	77.50 ± 0.25	54.69 ± 0.40	-
HaarPool	-	-	68.22± 0.86	-	-	-	89.98 ± 0.58	76.72 ± 0.60	53.03 ± 0.14	-
GMT (Ours)	82.19 ± 0.40	**80.01** ± 0.21	91.00 ± 0.82	**83.54** ± 0.78	**80.91** ± 0.41	**69.77** ± 0.67	95.14 ± 0.48	**78.43** ± 0.22	**55.14** ± 0.25	**83.37** ± 0.11

3.6.5 Experimental Details on Graph Generation

Common Implementation Details In the graph generation experiments, we replace the graph embedding function $f(G)$ from existing graph generation models to the proposed Graph Multiset Transformer (GMT), to evaluate the applicability of our model on generation tasks, as described in the subsection 3.4.3 of the main paper. As baselines, we first use the original models with their implementations. Specifically, we use the MolGAN[1] [116] for molecule generation, and Graph Logic Network (GLN)[2] [117] for retrosynthesis. For both experiments, we directly follow the experimental details of original papers [116, 117] for a fair comparison. Furthermore, to compare our models with another strong pooling method, we use the MinCutPool [83] as an additional baseline for generation tasks, since it shows the best performance among baselines in the previous two classification and reconstruction tasks.

For MinCutPool, since it cannot directly compress the all n nodes into the 1 cluster to represent the entire graph, we use the following trick to replace the simple pooling operation (e.g. sum or mean) with it. We first condense the graph into the k clusters ($k = 4$) using one MinCutPool layer, and then average the condensed nodes to get a single representation of the given graph. However, our proposed Graph Multiset Transformer (GMT) can directly compress the all n nodes into the 1 node with one learnable seed vector, by using the single GMPool_1 block. In other words, we use the one GMPool_1 to represent the entire graph by replacing their simple pooling (e.g. sum or mean), in which we use the following softmax activation function for computing attention weights:

$$w(\boldsymbol{Q}\boldsymbol{K}^T)_{i,j} = \frac{\exp(\boldsymbol{Q}_i \boldsymbol{K}_j^T)}{\sum_{k=1}^{n} \exp(\boldsymbol{Q}_i \boldsymbol{K}_k^T)}. \tag{3.14}$$

Implementation Details on Molecule Generation For the molecule generation experiment with the MolGAN, we replace the average pooling in the discriminator with GMPool_1. We use the QM9 dataset [121] following the original MolGAN paper [116]. To evaluate the models, we report the validity of 13,319 generated molecules at the early stage of the MolGAN training, over 4 different runs. As depicted in Figure 3.8 of the main paper, each solid curve indicates the average validity of each model with 4 different runs, and the shaded area indicates the half of the standard deviation.

[1] https://github.com/yongqyu/MolGAN-pytorch
[2] https://github.com/Hanjun-Dai/GLN

Table 3.6: Quantitative results of the graph reconstruction task on reconstructing the node features and the adjacency matrix for synthetic graphs, with two different minimization objectives and error calculation metrics: $\boldsymbol{X} - \boldsymbol{X}^{rec}$ and $\boldsymbol{A} - \boldsymbol{A}^{rec}$. * indicates the model without using adjacency normalization.

Data:	Grid Graph				Ring Graph			
Objective:	$\min\|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$		$\min\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$		$\min\|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$		$\min\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$	
Error Calculation:	$\|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$	$\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$	$\|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$	$\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$	$\|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$	$\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$	$\|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$	$\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$
DiffPool	0.0833	12110194	0.3908	0.0856	0.0032	617.7706	0.6208	0.0948
MinCutPool	0.0001	0.0092	1.2883	0.0051	0.0005	0.0424	0.5026	0.0128
MinCutPool*	0.0002	201.7619	2.0261	0.0616	0.0003	68.23	0.5211	0.0725
GMT (Ours)	0.0001	0.0102	0.2353	0.0084	0.0000	0.0331	0.5475	0.0324

Implementation Details on Retrosynthesis For the retrosynthesis experiment with the Graph Logic Network (GLN), we replace the average pooling in the template and subgraph encoding functions with GMPool_1. We use the USPTO-50k dataset following the original paper [117]. For an evaluation metric, we use the Top-k accuracy for both reaction class is not given and given cases, following the original paper [117]. We reproduce all results in Table 3.3 with published codes from the original paper.

3.6.6 Additional Experimental Results

Validation Results on Graph Classification We additionally provide the graph classification results on validation sets. As shown in Table 3.4, the proposed GMT outperforms most baselines, or achieves comparable performances to the best baseline results even in the validation sets. While validation results can not directly measure the generalization performance of the model for unseen data, these results further confirm that our method is powerful enough, compared to baselines. Regarding the results of test sets on the graph classification task, please see Table 3.1 in the main paper.

Leaderboard Results on Graph Classification For a fair comparison, we experiment with all baselines and our models in the same setting, as described in the implementation details of Appendix 3.6.3. Specifically, we average the results over 10 different runs with the same hidden dimension (128, while leaderboard uses 300), and the same number of message-passing layers (3, while leaderboard uses 5) with 10 different seeds for all models. Therefore, the reproduced results can be slightly different from the leaderboard results, as shown in Table 3.5, since the leaderboard uses different hyper-parameters with different random seeds (See Hu et al. [109] for more details). However, our reproduction results are almost the same as the leaderboard results, and sometimes outperform the leaderboard results (See the GCN results for the HIV dataset in Table 3.5). Therefore, while we conduct all experiments under the same setting for a fair comparison, where specific hyperparameter choices are slightly different from the leaderboard setting, these results indicate that there is no significant difference between reproduced and leaderboard results.

Table 3.5: Graph classification results for OGB test datasets with standard deviations.

	Model	**HIV**	**Tox21**
Leaderboard	GCN	76.06 ± 0.97	75.29 ± 0.69
	GIN	75.58 ± 1.40	74.91 ± 0.51
Reproduced	GCN	76.81 ± 1.01	75.04 ± 0.80
	GIN	75.95 ± 1.35	73.27 ± 0.84
Ours	GMT	**77.56** ± 1.25	**77.30** ± 0.59

Quantitative Results on Graph Reconstruction for Synthetic Graphs While we conduct experiments on reconstructing node features on the given graph, to quantify the retained information on the condensed nodes after pooling (See Section 3.4.2 for experiments on the graph reconstruction task), we further reconstruct the adjacency matrix to see if the pooling layer can also condense the adjacency structure without loss of information. The learning objective to minimize the discrepancy between the

original adjacency matrix $\boldsymbol{A}$ and the reconstructed adjacency matrix $\boldsymbol{A}^{rec}$ with a cluster assignment matrix $\boldsymbol{C} \in \mathbb{R}^{n \times k}$ is defined as follows: $\min\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$, where $\boldsymbol{A}^{rec} = \boldsymbol{C}\boldsymbol{A}^{pool}\boldsymbol{C}^{T}$.

Then we design the following two experiments. First, pooling layers are trained to minimize the objective in Section 3.4.2: $\min\|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$. After that, we measure the discrepancy between the original and the reconstructed node features: $\|\boldsymbol{X} - \boldsymbol{X}^{rec}\|$, and also measure the discrepancy between the original and the reconstructed adjacency matrix: $\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$. Second, pooling layers are trained to minimize the objective described in the previous paragraph: $\min\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$, and then we measure the aforementioned two discrepancies in the same way.

We experiment with synthetic grid and ring graphs, illustrated in Figure 3.10. Table 3.6 shows that the error is large when the objective and the error metric are different, which indicates that there is a high discrepancy between the required information for condensing node and the required information for condensing adjacency matrix. In other words, the compression for node and the compression for adjacency matrix might be differently performed to reconstruct the whole graph information.

Also, Table 3.6 shows that there are some cases where there is no significant difference in the calculated adjacency error ($\|\boldsymbol{A} - \boldsymbol{A}^{rec}\|$), when minimizing nodes discrepancies and minimizing adjacency discrepancies (See 0.0331 and 0.0324 for the proposed GMT on the Ring Graph). Furthermore, calculated errors for the adjacency matrix when minimizing adjacency discrepancies are generally larger than the calculated errors for node features when minimizing nodes discrepancies. These results indicate that the adjacency matrix is difficult to reconstruct after pooling. This might be because the reconstructed adjacency matrix should be further transformed from continuous values to discrete values (0 or 1 for the undirected simple graph), while the reconstructed node features can be directly represented as continuous values. We leave further reconstructing adjacency matrices and visualizing them as a future work.

Additional Examples for Molecule Reconstruction We visualize the additional examples for molecule reconstruction on the ZINC dataset in Figure 3.11. Molecules on the left side indicate the original molecule, where the transparent color denotes the assigned cluster for each node, which is obtained by the cluster assignment matrix $\boldsymbol{C}$ with node (atom) representations in a graph (molecule) (See Proposition 5 for more detail on generating the cluster assignment matrix). Also, molecules on the right side indicate the reconstructed molecules with failure cases denoted as a red dotted circle.

As visualized in Figure 3.11, we can see that the same atom or the similarly connected atoms obtain the same cluster (color). For example, the atom type O mostly obtains the yellow cluster, and the atom type F obtains the green cluster. Furthermore, ring-shaped substructures that do not contain O or N mostly receive the blue cluster, whereas ring-shaped substructures that contain O and N receive the green and yellow clusters respectively.

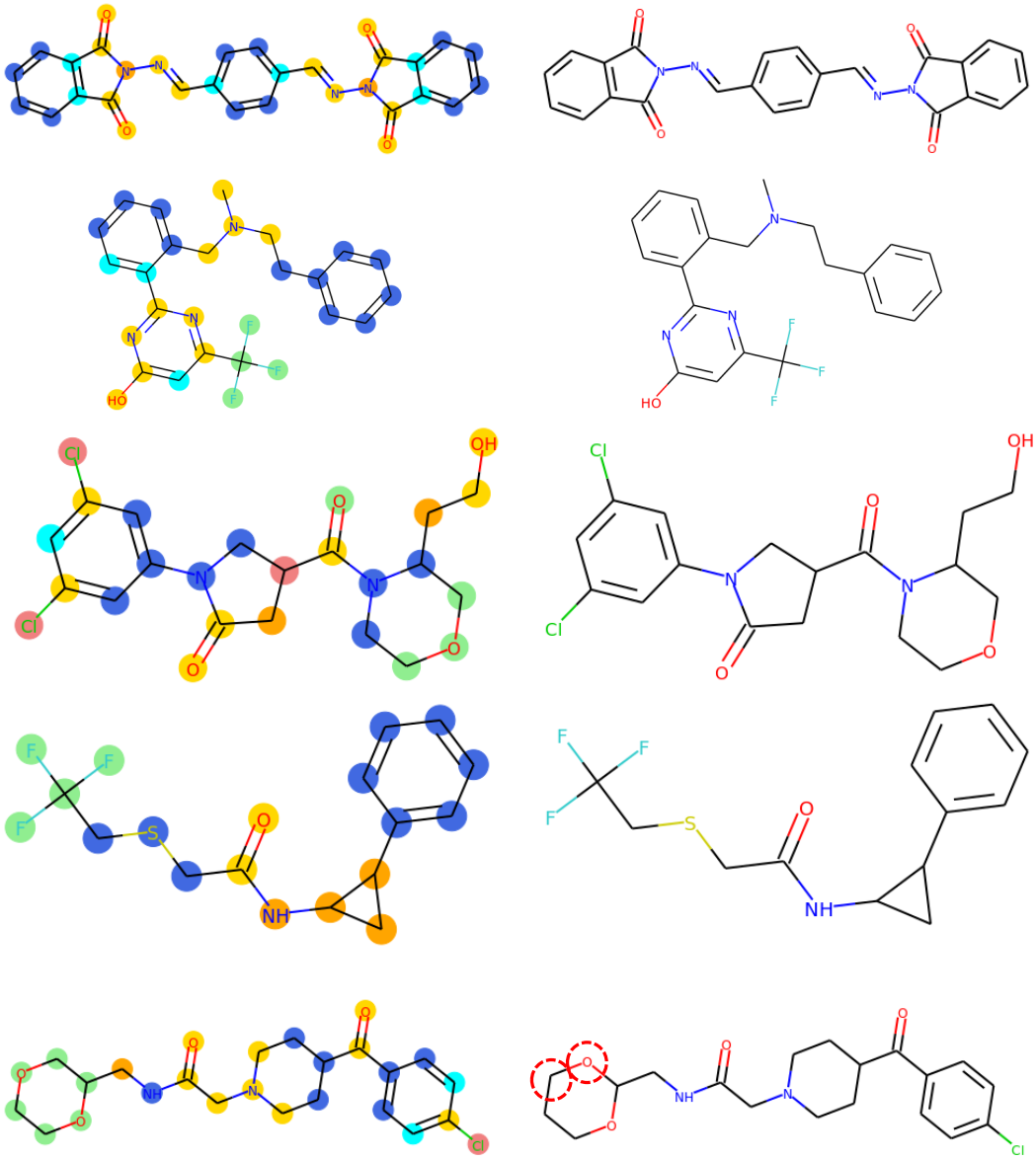

Figure 3.11: **Molecule Reconstruction Examples (Left):** Original molecules with the assigned cluster on each node represented as color, where cluster is generated from *Graph Multiset Pooling* (GMPool). **(Right):** Reconstructed molecules. Red dotted circle indicates the incorrect atom prediction.

Chapter 4. Concluding Remark

In this thesis, we focus on the current limitations of node-level graph neural networks (GNNs) for representing unseen nodes and entire graphs. Specifically, in Chapter 2, we aim at accurately representing unseen nodes that are not seen during training but appear at test time. To this end, we first formally define a problem of unseen nodes on a real-world evolving graph, which we refer to as few-shot out-of-graph link prediction, and then propose the transductive meta-learning framework that simulates unseen entities during training to handle newly emerged entities at test time. On the other hand, in Chapter 3, we aim at obtaining entire graph representations that are distinct across different graphs. To this end, we propose a transformer-based graph pooling method that not only captures interaction among all nodes, but also is approximated to be at most as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test. We evaluate the performance of our models on tasks for unseen nodes and entire graphs, and show that the proposed methods can accurately represent both of them, while node-level GNNs fail to do so.

We believe our models largely contribute to learning unobserved nodes and whole graphs more accurately, and bring huge potential impacts in application to real-world graphs, for instance, evolving graphs – knowledge graph, social network, and communication network – that grow with new nodes over time. Also, ours could be broadly used to model the entire graph, such as representing a molecular graph, analyzing a social network, and detecting an attack in a cybernetwork. However, there are still remaining challenges that hurt the performance of GNNs on graph-related tasks. First, in a data-level perspective, graphs, that are automatically constructed from the machine or even manually annotated from the human, are inaccurate (i.e., there are missing links, and even available connections between nodes are often incorrect). Thus, we further need to consider the accurateness of the predicted link, and even develop the method for cleaning out the noise on the constructed graph. Also, in terms of machine learning schemes, we should accurately reflect all sources of information on the graph, such as features of both nodes and edges, and their underlying structural roles. Furthermore, in an application-level perspective, it is required to explore the data – whose underlying structure is a graph, yet getting less attention to model with GNNs, such as neural networks – and the corresponding method for exploiting such particular data. For the rest, if an obtained graph representation is satisfactorily considered to be accurate, methods of fusing the graph knowledge into other domains (e.g., computer vision and natural language understanding) would be hopeful to further develop.

Bibliography

[1] Yongji Wu, Defu Lian, Yiheng Xu, Le Wu, and Enhong Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1054–1061. AAAI Press, 2020.

[2] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 8459–8468. PMLR, 2020.

[3] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, pages 593–607, 2018.

[4] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint*, arXiv:1812.08434, 2018.

[5] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[6] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[7] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 5171–5181, 2018.

[8] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, Peter W. Battaglia, Vishal Gupta, Ang Li, Zhongwen Xu, Alvaro Sanchez-Gonzalez, Yujia Li, and Petar Velickovic. ETA prediction with graph neural networks in google maps. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 3767–3776. ACM, 2021.

[9] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 974–983. ACM, 2018.

[10] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[11] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3835–3845. PMLR, 2019.

[12] B. Yu. Weisfeiler and A. A. Leman. Reduction of a graph to a canonical form and an algebra arising during this reduction. 1968.

[13] Jinheon Baek, Dong Bok Lee, and Sung Ju Hwang. Learning to extrapolate knowledge: Transductive few-shot out-of-graph link prediction. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3837–3845, 2016.

[15] Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3060–3067, 2019.

[16] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1506–1515, 2017.

[17] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6530–6539, 2017.

[18] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. In *Bioinformatics*, page 34(13):i457–i466, 2018.

[19] Chence Shi, Minkai Xu, Hongyu Guo, Ming Zhang, and Jian Tang. A graph to graphs framework for retrosynthesis prediction. *arXiv preprint arXiv:2003.12725*, 2020.

[20] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250, 2008.

[21] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[22] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 777–782, 2013.

[23] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795, 2013.

[24] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[25] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1811–1818, 2018.

[26] Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Q. Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 327–333, 2018.

[27] Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4710–4723, 2019.

[28] PeiFeng Wang, Jialong Han, Chenliang Li, and Rong Pan. Logic attention based neighborhood aggregation for inductive knowledge graph embedding. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 7152–7159, 2019.

[29] Baoxu Shi and Tim Weninger. Open-world knowledge graph completion. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1957–1964, 2018.

[30] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 564–573, 2017.

[31] Jae Yong Ryu, Hyun Uk Kim, and Sang Yup Lee. Deep learning improves prediction of drug–drug and drug–food interactions. *Proceedings of the National Academy of Sciences*, 115(18):E4304–E4311, 2018. ISSN 0027-8424. doi: 10.1073/pnas.1803294115. URL https://www.pnas.org/content/115/18/E4304.

[32] Marinka Zitnik, Rok Sosič, Sagar Maheshwari, and Jure Leskovec. BioSNAP Datasets: Stanford biomedical network dataset collection. http://snap.stanford.edu/biodata, August 2018.

[33] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, July 2005.

[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[35] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1024–1034, 2017.

[36] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[37] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Unsupervised universal self-attention network for graph classification. *arXiv preprint arXiv:1909.11855*, 2019.

[38] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

[39] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[40] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3630–3638, 2016.

[41] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 4077–4087, 2017.

[42] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1126–1135, 2017.

[43] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[44] Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[45] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. Learning to propagate for graph meta-learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 1037–1048, 2019.

[46] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 2357–2360, 2019.

[47] Kaize Ding, Jianling Wang, Jundong Li, Kai Shu, Chenghao Liu, and Huan Liu. Graph prototypical networks for few-shot learning on attributed networks. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 295–304. ACM, 2020.

[48] Lin Lan, Pinghui Wang, Xuefeng Du, Kaikai Song, Jing Tao, and Xiaohong Guan. Node classification on graphs with few-shot novel labels via meta transformed network embedding. *arXiv preprint arXiv:2007.02914*, 2020.

[49] Avishek Joey Bose, Ankit Jain, Piero Molino, and William L. Hamilton. Meta-graph: Few shot link prediction via meta learning. *arXiv preprint arXiv:1912.09867*, 2019.

[50] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1112–1119, 2014.

[51] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2071–2080, 2016.

[52] Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. One-shot relational learning for knowledge graphs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1980–1990, 2018.

[53] Mingyang Chen, Wen Zhang, Wei Zhang, Qiang Chen, and Huajun Chen. Meta relational learning for few-shot link prediction in knowledge graphs. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on*

Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019, pages 4216–4225, 2019.

[54] Chuxu Zhang, Huaxiu Yao, Chao Huang, Meng Jiang, Zhenhui Li, and Nitesh V. Chawla. Few-shot knowledge graph completion. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3041–3048. AAAI Press, 2020.

[55] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. Representation learning of knowledge graphs with entity descriptions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2659–2665, 2016.

[56] Zihao Wang, Kwun Ping Lai, Piji Li, Lidong Bing, and Wai Lam. Tackling long-tailed relations and uncommon entities in knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 250–260, 2019.

[57] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1802–1808, 2017.

[58] Marjan Albooyeh, Rishab Goel, and Seyed Mehran Kazemi. Out-of-sample representation learning for multi-relational graphs. *arXiv preprint arXiv:2004.13230*, 2020.

[59] Tengfei Ma, Junyuan Shang, Cao Xiao, and Jimeng Sun. GENN: predicting correlated drug-drug interactions with graph energy neural networks. *arXiv preprint arXiv:1910.02107*, 2019.

[60] B. Wang, A. Mezlini, F. Demir, M. Fiume, Z. Tu, M. Brudno, and A. Goldenberg. Similarity network fusion for aggregating data types on a genomic scale. In *Nature Methods*, page 11:333–337, 2014.

[61] P. Zhang, F. Wang, J. Hu, and R. Sorrentino. Label propagation prediction of drug-drug interaction. In *Scientific reports*, 2015.

[62] Wen Zhang, Yanlin Chen, Feng Liu, Fei Luo, Gang Tian, and Xiaohong Li. Predicting potential drug-drug interactions by integrating chemical, biological, phenotypic and network data. *BMC Bioinformatics*, 18(1):18:1–18:12, 2017.

[63] Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[64] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3483–3491, 2015.

[65] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1050–1059. JMLR.org, 2016.

[66] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Learning to model the tail. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 7029–7039, 2017.

[67] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1499–1509, 2015.

[68] Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr., Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil A. Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2302–2310, 2015.

[69] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[70] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, 2011.

[71] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha P. Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 5516–5522. Association for Computational Linguistics, 2020.

[72] David S. Wishart, Craig Knox, Anchi Guo, Dean Cheng, Savita Shrivastava, Dan Tzur, Bijaya Gautam, and Murtaza Hassanali. Drugbank: a knowledgebase for drugs, drug actions and drug targets. *Nucleic Acids Res.*, 36(Database-Issue):901–906, 2008.

[73] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[74] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[75] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[76] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[77] Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D. Yoo. Edge-labeling graph neural network for few-shot learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 11–20, 2019.

[78] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[79] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4438–4445. AAAI Press, 2018.

[80] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 4805–4815, 2018.

[81] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1993–2001, 2016.

[82] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 2019.

[83] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. *arXiv preprint*, arXiv:1907.00481, 2019.

[84] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019.

[85] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2083–2092. PMLR, 2019.

[86] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7134–7143. PMLR, 2019.

[87] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. GROVER: self-supervised message passing transformer on large-scale molecular data. *arXiv preprint arXiv:2007.02835*, 2020.

[88] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2224–2232, 2015.

[89] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2702–2711, 2016.

[90] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 29–38. IEEE Computer Society, 2017.

[91] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 723–731. ACM, 2019.

[92] Yu Guang Wang, Ming Li, Zheng Ma, Guido Montúfar, Xiaosheng Zhuang, and Yanan Fan. Haar-pooling: Graph pooling with compressive haar basis. *arXiv preprint arXiv:1909.11580*, 2019.

[93] Hao Yuan and Shuiwang Ji. Structpool: Structured graph pooling via conditional random fields. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[94] Ekagra Ranjan, Soumya Sanyal, and Partha P. Talukdar. ASAP: adaptive structure aware pooling for learning hierarchical graph representations. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5470–5477. AAAI Press, 2020.

[95] Frederik Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

[96] Amir Hosein Khas Ahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. Memory-based graph networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[97] Jia Li, Yu Rong, Hong Cheng, Helen Meng, Wen-bing Huang, and Junzhou Huang. Semi-supervised graph classification: A hierarchical graph perspective. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 972–982. ACM, 2019.

[98] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[99] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 77–85. IEEE Computer Society, 2017.

[100] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas J. Guibas. GSPN: generative shape proposal network for 3d instance segmentation in point cloud. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 3947–3956. Computer Vision Foundation / IEEE, 2019.

[101] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3391–3401, 2017.

[102] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 3744–3753, 2019.

[103] Catalina Cangea, Petar Velickovic, Nikola Jovanovic, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint*, arXiv:1811.01287, 2018.

[104] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[105] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint*, arXiv:1607.06450, 2016.

[106] Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A. Osborne. On the limitations of representing functions on sets. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6487–6494. PMLR, 2019.

[107] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[108] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint*, arXiv:2007.08663, 2020.

[109] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint*, arXiv:2005.00687, 2020.

[110] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[111] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, 2011.

[112] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[113] Diego P. P. Mesquita, Amauri H. Souza Jr., and Samuel Kaski. Rethinking pooling in graph neural networks. *arXiv preprint arXiv:2010.11418*, 2020.

[114] P. Erdős and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960.

[115] John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. ZINC: A free tool to discover chemistry for biology. *J. Chem. Inf. Model.*, 52(7):1757–1768, 2012.

[116] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint*, arXiv:1805.11973, 2018.

[117] Hanjun Dai, Chengtao Li, Connor W. Coley, Bo Dai, and Le Song. Retrosynthesis prediction with conditional graph logic network. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8870–8880, 2019.

[118] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2014–2023. JMLR.org, 2016.

[119] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5449–5458. PMLR, 2018.

[120] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint*, arXiv:2003.00982, 2020.

[121] R. Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

Acknowledgment

I first would like to thank my advisor, Prof. Sung Ju Hwang, for providing insightful and valuable ideas with a great passion for our research. Without his supports, all my research accomplishments would not have been possible. I always look up to him as a remarkable scientist, and want to improve my research skill set like him. I also would like to thank my thesis committee members Prof. Eunho Yang, and Prof. Kijung Shin for their constructive and thoughtful comments on my research.

My sincere thanks also go out to my labmates, especially to my coauthors and collaborators, who have influenced my daily growth. Thanks to Dong Bok for guiding me when I first came into our lab, to Minki for sharing valuable wisdom during any discussion that makes our research more solid, to Jaehyeong for incredibly hard work that inspires me a lot with great friendship, to Dongki for sharing attitudes of never giving up, and to Seul for sharing insight on biochemistry. Also, I want to thank Wonyong, Hayeon, Gun, and Eunyoung, with whom I got an opportunity to learn how to collaborate together in large-scale research. Most of all, every labmate – from current: Jaehong, Haebeom, Jin Myung, Hayeon, Bruno, Minseon, Taewook, Wonyong, Byungjoo, Minki, Dongbok, Jeff, Jaehyeong, Minyoung, Seanie, Dongchan, Seul, Hyunsu, Geon, Sunil, Dongki, and Jongha, to former: Moonsu, Hyewon, Eunyoung, Jaewoong, Divyam, Wuhyun, and Jeongun – makes my life in a school an enjoyable and unforgettable one. Thank you for giving generous help when I needed it.

I also want to thank external collaborators. Thanks to DMIS lab members in my undergraduate school, especially to Prof. Jaewoo Kang for providing me the chance to conduct my first research but also thoughtfully supporting it, to Hyunjae and Yookyung for our collaboration work even when I left there, and to Donghyeon and Buru for mentoring my youngest time of research journey. I am also thankful to Prof. Jong C. Park for providing constructive feedback, as well as to Soyeong, Chaehun and Sukmin for sharing valuable ideas with intense discussions, on our research collaboration.

Before and even after starting my academic journey, there are great teachers, mentors and programs, that support my studies of computer science and engineering, and encourage me to pursue a Master's degree of artificial intelligence. Thanks to the Samsung Dream Scholarship Foundation for supporting many aspects of my studies toward a researcher, to the Microsoft Student Partner program with Mr. Minsoo Bae and Ms. Eunji Kim from Microsoft for providing me the chance of learning computer science on a global scale, to the SW Maestro program from which I decide to study artificial intelligence, to Prof. Heejo Lee for guiding me in the programs of KUICS and Inc0gnito, therein always asking about what I should pursue in the future, to Mr. Kwangjae Jang for encouraging me to choose the major of computer science and engineering.

Beyond my academic life, I owe my wonderful time to my friends outside the lab too. Although there are too many to list individually, thanks to all my friends for helping to take the stress out of my graduate school, and cheering up my academic journey – especially friends from the KU Honam community, to the KU Computer Science and Engineering Department, to the KU WoonWha volunteering club, to the Samsung Dream Scholarship Foundation, to the Microsoft Student Partner program. Special thanks to Soyeong for being a best friend, as well as enriching my life.

Last but not least, I would like to thank my family – my father Cheonseok Baek, my mother Miran Jeong, and my younger sister Jina Baek – for their endless support and love.

Curriculum Vitae

Name : Jinheon Baek

Date of Birth : July 16, 1997

Educations

2020. 3. – 2022. 2. M.S. in Artificial Intelligence, KAIST

2016. 3. – 2020. 2. B.S. in Computer Science and Engineering, Korea University
B.E. in Software Technology and Enterprise Program, Korea University

Publications

1. M. Kang*, **J. Baek***, and S. J. Hwang. KALA: Knowledge-Augmented Language Model Adaptation. In preparation. (*: equal contribution)
2. S. Jeong, **J. Baek**, S. Cho, S. J. Hwang, and J. C. Park. Augmenting Document Representations for Dense Retrieval with Interpolation and Perturbation. In preparation.
3. J. Jo*, **J. Baek***, S. Lee*, D. Kim, M. Kang, and S. J. Hwang. Edge Representation Learning with Hypergraphs. Conference on Neural Information Processing Systems **(NeurIPS), 2021**. (*: equal contribution)
4. W. Jeong*, H. Lee*, G. Park*, E. Hyung, **J. Baek**, and S. J. Hwang. Task-Adaptive Neural Network Retrieval with Meta-Contrastive Learning. Conference on Neural Information Processing Systems **(NeurIPS), 2021**. **(Spotlight Presentation)** (*: equal contribution)
5. S. Jeong, **J. Baek**, C. Park, and J. C. Park. Unsupervised Document Expansion for Information Retrieval with Stochastic Text Generation. Scholarly Document Processing at Conference of the North American Chapter of the Association for Computational Linguistics **(SDP@NAACL), 2021**. **(Oral Presentation)**
6. **J. Baek***, M. Kang*, and S. J. Hwang. Accurate Learning of Graph Representations with Graph Multiset Pooling. International Conference on Learning Representations **(ICLR), 2021**. (*: equal contribution)
7. H. Kim*, Y. Koh*, **J. Baek**, and J. Kang. Exploring The Spatial Reasoning Ability of Neural Models in Human IQ Tests. **Neural Networks, 2021**. (*: equal contribution)
8. **J. Baek**, D. B. Lee, and S. J. Hwang. Learning to Extrapolate Knowledge: Transductive Few-shot Out-of-Graph Link Prediction. Conference on Neural Information Processing Systems **(NeurIPS), 2020**. (*: equal contribution)