TaxoExpan: Self-supervised Taxonomy Expansion with Position-Enhanced Graph Neural Network

Jiaming Shen[1*], Zhihong Shen[2], Chenyan Xiong[2], Chi Wang[2], Kuansan Wang[2], Jiawei Han[1]
[1]University of Illinois at Urbana-Champaign, IL, USA [2] Microsoft Research, WA, USA
[1]{js2, hanj}@illinois.edu [2]{Zhihosh, Chenyan.xiong, Wang.chi, kuansanw}@microsoft.com

ABSTRACT

Taxonomies consist of machine-interpretable semantics and provide valuable knowledge for many web applications. For example, online retailers (*e.g.*, Amazon and eBay) use taxonomies for product recommendation, and web search engines (*e.g.*, Google and Bing) leverage taxonomies to enhance query understanding. Enormous efforts have been made on constructing taxonomies either manually or semi-automatically. However, with the fast-growing volume of web content, existing taxonomies will become outdated and fail to capture emerging knowledge. Therefore, in many applications, dynamic expansions of an existing taxonomy are in great demand. In this paper, we study how to expand an existing taxonomy by adding a set of new concepts. We propose a novel self-supervised framework, named TaxoExpan, which automatically generates a set of ⟨query concept, anchor concept⟩ pairs from the existing taxonomy as training data. Using such *self-supervision* data, TaxoExpan learns a model to predict whether a query concept is the direct hyponym of an anchor concept. We develop two innovative techniques in TaxoExpan: (1) a position-enhanced graph neural network that encodes the local structure of an anchor concept in the existing taxonomy, and (2) a noise-robust training objective that enables the learned model to be insensitive to the label noise in the self-supervision data. Extensive experiments on three large-scale datasets from different domains demonstrate both the effectiveness and the efficiency of TaxoExpan for taxonomy expansion.

KEYWORDS

Taxonomy Expansion; Self-supervised Learning

1 INTRODUCTION

Taxonomies have been fundamental to organizing knowledge for centuries [39]. In today's Web, taxonomies provide valuable knowledge to support many applications such as query understanding [14], content browsing [46], personalized recommendation [15, 55], and web search [24, 45]. For example, many online retailers (*e.g.*, eBay and Amazon) organize products into categories of different granularities, so that customers can easily search and navigate this category taxonomy to find the items they want to purchase. In addition, web search engines (*e.g.*, Google and Bing) leverage a taxonomy to better understand user queries and improve the search quality.

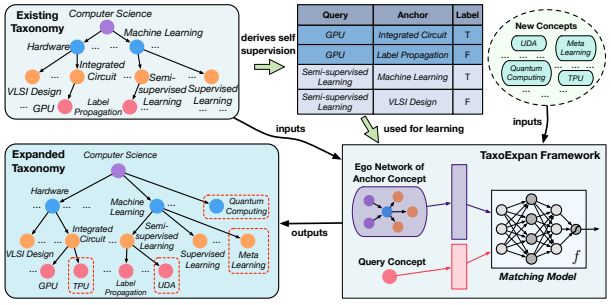

Figure 1: An example of expanding one computer science field-of-study taxonomy to include new concepts such as "*Quantum Computing*", "*Meta Learning*", and "*TPU*".

Existing taxonomies are mostly constructed by human experts or in a crowdsourcing manner. Such manual curations are time-consuming, labor-intensive, and rarely complete. To reduce the human efforts, many automatic taxonomy construction methods [26, 34, 52] are proposed. They first identify "is-A" relations (*e.g.*, "*iPad*" is an "*Electronics*") using textual patterns [13, 31] or distributional similarities [2, 37], and then organize extracted concept pairs into a directed acyclic graph (DAG) as the output taxonomy [7, 11, 20]. As the web contents and human knowledge are constantly growing, people need to expand an existing taxonomy to include new emerging concepts. Most of previous methods, however, construct a taxonomy entirely *from scratch* and thus when we add new concepts, we have to re-run the entire taxonomy construction process. Although being intuitive, this approach has several limitations. First, many taxonomies have a top-level design provided by domain experts and such design shall be preserved. Second, a newly constructed taxonomy may not be consistent with the old one, which can lead to instabilities of its dependent downstream applications. Finally, as targeting the scenario of building taxonomy from scratch, most previous methods are unsupervised and cannot leverage signals from the existing taxonomy to construct a new one.

In this paper, we study the *taxonomy expansion* task: given an existing taxonomy and a set of new emerging concepts, we aim to automatically expand the taxonomy to incorporate these new concepts (without changing the existing relations in the given taxonomy).[1] Figure 1 shows an example where a taxonomy in computer science domain is expanded to include new subfields (*e.g.*, "*Quantum Computing*") and new techniques (*e.g.*, "*Meta Learning*" and "*UDA*"). Some previous studies [17, 18, 32] attempt this task by using an

*This work is done while interning at Microsoft Research.

WWW '20, April 20–24,2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380132

[1]We recognize that the modification of an existing taxonomy is necessary in some cases. However, it happens much less frequently and requires high cautiousness from human curator. Therefore, we leave it out of the scope of automation.

additional set of labeled concepts with their true insertion positions in the existing taxonomy. However, such labeled data are usually small and thus forbid us from learning a more powerful model that captures the subsumption semantics in the existing taxonomy.

We propose a novel framework named TaxoExpan to tackle the lack-of-supervision challenge. TaxoExpan formulates a taxonomy as a directed acyclic graph (DAG), automatically generates pseudo-training data from the existing taxonomy, and uses them to learn a matching model for expanding a given taxonomy. Specifically, we view each concept in the existing taxonomy as a *query* and one of its parent concepts as an *anchor*. This gives us a set of positive ⟨query concept, anchor concept⟩ pairs. Then, we generate negative pairs by sampling those concepts that are neither the descendants nor the direct parents of the query concept in the existing taxonomy. In Figure 1, for example, the ⟨"*GPU*", "*Integrated Circuit*"⟩ is a positive pair and ⟨"*GPU*", "*Label Propagation*"⟩ is a negative pair. We refer to these training pairs as *self-supervision* data, because they are procedurally generated from the existing taxonomy and no human curation is involved.

To make the best use of above self-supervision data, we develop two novel techniques in TaxoExpan. The first one is a position-enhanced graph neural network (GNN) which encodes the local structure of an anchor concept using its ego network (egonet) in the existing taxonomy. If we view this anchor concept as the "parent" of the query concept, this ego network includes the potential "siblings" and "grand parents" of the query concept. We apply graph neural networks (GNNs) to model this ego network. However, regular GNNs fail to distinguish nodes with different relative positions to the query (*i.e.*, some nodes are grand parents of the query while the others are siblings of the query). To address this limitation, we present a simple but effective enhancement to inject such position information into GNNs using position embedding. We show that such embedding can be easily integrated with existing GNN architectures (*e.g.*, GCN [19] and GAT [43]) and significantly boosts the prediction performance. The second technique is a new noise-robust training scheme based on the InfoNCE loss [41]. Instead of predicting whether each individual ⟨query concept, anchor concept⟩ pair is positive or not, we first group all pairs sharing the same query concept into a single training instance and learn a model to select the positive pair among other negative ones from the group. We show that such training scheme is robust to the label noise and leads to performance gains.

We test the effectiveness of TaxoExpan framework on three real-world taxonomies from different domains. Our results show that TaxoExpan can generate high-quality concept taxonomies in scientific domains and achieves state-of-the-art performance on the WordNet taxonomy expansion challenge [18].

Contributions. To summarize, our major contributions include: (1) a self-supervised framework that automatically expands existing taxonomies without manually labeled data; (2) an effective method for enhancing graph neural network by incorporating hierarchical positional information; (3) a new training objective that enables the learned model to be robust to label noises in self-supervision data; and (4) extensive experiments that verify both the effectiveness and the efficiency of TaxoExpan framework on three real-world large-scale taxonomies from different domains.

2 RELATED WORK

Taxonomy Construction and Expansion. Most existing taxonomy construction methods focus on building the *entire* taxonomy by first extracting hypernym-hyponym pairs and then organizing all hypernymy relations into a tree or DAG structure. For the first hypernymy discovery step, methods fall into two categories: (1) *pattern-based* methods which leverage pre-defined patterns [13, 16, 29] to extract hypernymy relations from a corpus, and (2) *distributional* methods which calculate pairwise term similarity metrics based on term embeddings [22, 25, 36] and use them to predict whether two terms hold the hypernymy relation. For the second hypernymy organization step, most methods formulate it as a graph optimization problem. They first build a noisy hypernymy graph using hypernymy pairs extracted and then derive the output taxonomy as a particular tree or DAG structure (*e.g.*, maximum spanning tree [4], and minimum-cost flow [11]). Finally, there are some methods that leverage entity set expansion techniques [33, 54] to incrementally construct a taxonomy either from scratch or from a tiny seed taxonomy.

In many real-world applications, some existing taxonomies may have already been laboriously curated by experts [9, 23] or via crowdsourcing [27], and are deployed in online systems. Instead of constructing the entire taxonomy from scratch, these applications demand the feature of expanding an existing taxonomy dynamically. There exist some studies on expanding WordNet with named entities from Wikipedia [40] or domain-specific concepts from different corpora [3, 10, 17]. Task 14 of SemEval 2016 challenge [18] is specifically setup to enrich WordNet with domain-specific concepts. One limitation of these approaches is that they depend on the synset structure unique to WordNet and thus cannot be easily generalized to other taxonomies.

To address the above limitation, more recent works try to develop methodologies for expanding a generic taxonomy. Wang *et al.* [44] design a hierarchical Dirichlet model to extend the category taxonomy in search engines using query logs. Plachouras *et al.* [30] learn paraphrase models on external paraphrase datasets and apply learned models to directly find paraphrases of concepts in the existing taxonomy. Vedula *et al.* [42] combine multiple features, some of which are retrieved from an external `Bing` Search API, into a ranking model to score candidate positions in terms of their matching scores with the query concept. Comparing with these methods, our TaxoExpan framework explicitly models the local structure around each candidate position, which boosts the quality of expanded taxonomy.

Graph Neural Network. Our work is also related to Graph Neural Network (GNN) which is a generic method of learning on graph-structure data. Many GNN architectures have been proposed to either learn individual node embeddings [12, 19, 43] for the node classification and the link prediction tasks or learn an entire graph representation [48, 53] for the graph classification task. In this work, we tackle the taxonomy expansion task with a fundamentally different formulation from previous tasks. We leverage some existing GNN architectures and enrich them with additional relative position information. Recently, You *et al.* [50] propose a method to add position information into GNN. Our methods are different from You *et al.*. They model the *absolute* position of a node in a full

graph without any particular reference points; while our technique captures the *relative* position of a node with respect to the query node. Finally, some work on graph generation [21, 49] involves a module to add a new node into a partially generated graph, which shares the similar goal as our model. However, such graph generation model typically requires fully labeled training data to learn from. To the best of our knowledge, this is the first study on how to expand an existing directed acyclic graph (as we model a taxonomy as a DAG) using self-supervised learning.

3 PROBLEM FORMULATION

In this section, we first define a taxonomy, then formulate our problem, and finally discuss the scope of our study.

Taxonomy. A taxonomy $\mathcal{T} = (\mathcal{N}, \mathcal{E})$ is a directed acyclic graph where each node $n \in \mathcal{N}$ represents a concept (*i.e.*, a word or a phrase) and each directed edge $\langle n_p, n_c \rangle \in \mathcal{E}$ indicates a relation expressing that concept n_p is the most specific concept that is more general than concept n_c. In other words, we refer to n_p as the "*parent*" of n_c and n_c as the "*child*" of n_p.

Problem Definition. The input of the *taxonomy expansion task* includes two parts: (1) an existing taxonomy $\mathcal{T}^0 = (\mathcal{N}^0, \mathcal{E}^0)$, and (2) a set of new concepts C. This new concept set can be either manually specified by users or automatically extracted from text corpora. Our goal is to expand the existing taxonomy $\mathcal{T}^0$ into a larger taxonomy $\mathcal{T} = (\mathcal{N}^0 \cup C, \mathcal{E}^0 \cup \mathcal{R})$, where $\mathcal{R}$ is a set of newly discovered relations each including one new concept $c \in C$.

Example 1. *Figure 1 shows an example of our problem. Given a field-of-study taxonomy $\mathcal{T}^0$ in the computer science domain and a set of new concepts $C = \{$"UDA", "Meta Learning", ...$\}$, we find each new concept's best position in $\mathcal{T}^0$ (*e.g.*, "UDA" under "Semi-supervised Learning" as well as "GPU" under "Integrated Circuit") and expand $\mathcal{T}^0$ to include those new concepts.*

Simplified Problem. A simplified version of the above problem is that we assume the input set of new concepts contains only one element (*i.e.*, $|C| = 1$), and we aim to find one single parent node of this new concept (*i.e.*, $|\mathcal{R}| = 1$). We discuss the connection between these two problem settings at the end of Section 4.1.

Discussion. In this work, we follow previous studies [1, 18, 42] and assume each concept in $\mathcal{N}^0 \cup C$ has an initial embedding vector learned from this concept's surface name, or if available, its definition sentences [32] and associated web pages [44]. We also note that our problem formulation assumes those relations in the existing taxonomy are not modified. We acknowledge that such modification is necessary in some cases, but it is much less frequent and requires high cautiousness from human curators. Therefore, we leave it out of the scope of automation in this study.

4 THE TAXOEXPAN FRAMEWORK

In this section, we first introduce our taxonomy model and expansion goal. Then, we elaborate how to represent a query concept and an insertion position (*i.e.*, an anchor concept), based on which we present our query-concept matching model. Finally, we discuss how to generate self-supervision data from the existing taxonomy and use them to train the TaxoExpan framework.

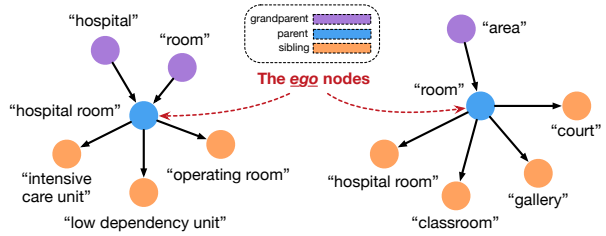

Figure 2: Two egonets correspond to two anchor concepts.

4.1 Taxonomy Model and Expansion Goal

A taxonomy $\mathcal{T}$ describes a hierarchical organization of concepts. These concepts form the node set $\mathcal{N}$ in $\mathcal{T}$. Mathematically, we model each node $n \in \mathcal{N}$ as a categorical random variable and the entire taxonomy $\mathcal{T}$ as a Bayesian network. We define the probability of a taxonomy $\mathcal{T}$ as the joint probability of node set $\mathcal{N}$ which can be further factorized into a set of conditional probabilities as follows:

$$\mathbf{P}(\mathcal{T}|\Theta) = \mathbf{P}(\mathcal{N}|\mathcal{T}, \Theta) = \prod_{i=1}^{|\mathcal{N}|} \mathbf{P}(n_i | parent_{\mathcal{T}}(n_i), \Theta),$$

where Θ is the set of model parameters and $parent_{\mathcal{T}}(n_i)$ is the set of n_i's parent node(s) in taxonomy $\mathcal{T}$.

Given learned model parameters Θ, an existing taxonomy $\mathcal{T}^0 = (\mathcal{N}^0, \mathcal{E}^0)$, and a set of new concepts C, we can ideally find the best taxonomy $\mathcal{T}^*$ by solving the following optimization problem:

$$\mathcal{T}^* = \arg\max_{\mathcal{T}} \mathbf{P}(\mathcal{T}|\Theta) = \arg\max_{\mathcal{T}} \sum_{i=1}^{|\mathcal{N}^0 \cup C|} \log \mathbf{P}(n_i | parent_{\mathcal{T}}(n_i), \Theta).$$

This naïve approach has two limitations. First, the search space of all possible taxonomies over the concept set $|\mathcal{N}^0 \cup C|$ is prohibitively large. Second, we cannot guarantee the structure of existing taxonomy $\mathcal{T}^0$ remains unchanged, which can be undesirable from the application point of view.

We address the above limitations by restricting the search space of our output taxonomy to be the exact expansion of the existing taxonomy $\mathcal{T}^0$. Specifically, we keep the parents of each existing taxonomy node $n \in \mathcal{N}^0$ unchanged and only try to find a *single* parent node of each new concept in C. As a result, we divide the above computationally intractable problem into the following set of $|C|$ tractable optimization problems:

$$a_i^* = \arg\max_{a_i \in \mathcal{N}^0} \log \mathbf{P}(n_i | a_i, \Theta), \quad \forall i \in \{1, 2, \ldots, |C|\}, \tag{1}$$

where a_i is the parent node of a new concept $n_i \in C$ and we refer to it as the "*anchor concept*".

Discussion. The above equation defines $|C|$ *independent* optimization problems and each problem aims to find one single parent of a new concept n_i. Therefore, we essentially reduce the more generic taxonomy expansion problem into $|C|$ independent simplified problems (c.f. Section 3) and tackle it by inserting new concepts *one-by-one* into the existing taxonomy. As a result of the above reduction, possible interactions among new concepts are ignored and we leave it to the future work. In the following sections, we continue to answer two keys questions: (1) how to model the conditional probability $\mathbf{P}(n_i|a_i, \Theta)$, and (2) how to learn model parameters Θ.

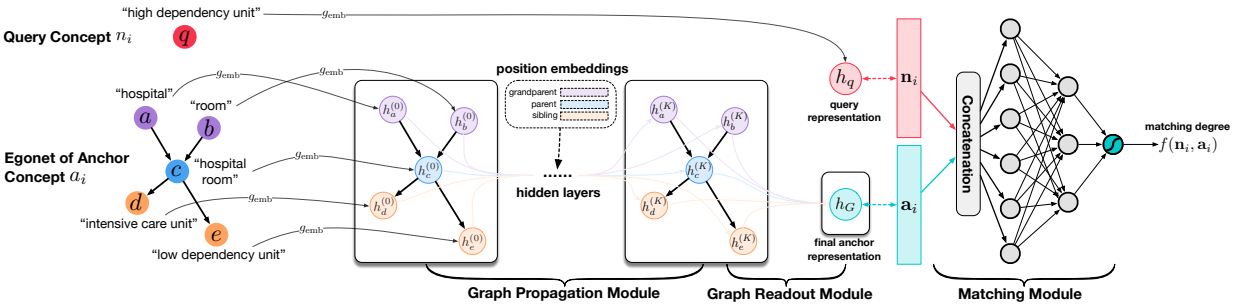

Figure 3: Overview of TaxoExpan framework. g_{emb} is an embedding model that provides query concept's initial feature vector h_q and the initial feature vector of each node in the egonet. The graph propagation module transforms initial feature vectors into better node representations based on which the graph readout module outputs the egonet embedding as the final anchor representation. Finally, a matching module inputs both query and anchor representations and outputs their matching score.

4.2 Modeling Query-Anchor Matching

We model the matching score between a query concept n_i and an anchor concept a_i by projecting them into a vector space and calculating matching scores using their vectorized representations. We show the entire model architecture of TaxoExpan in Figure 3.

4.2.1 *Representing Query Concept.*

In this study, we assume each query concept has an *initial feature vector* learned based on some text associated with this concept. Such text can be as simple as the concept surface name, or in some prior studies [18, 44], the definition sentences and clicked web pages about the concept. We represent each query concept n_i using its initial feature vector denoted as $\mathbf{n}_i$. We will discuss how to obtain such initial feature vectors using embedding learning methods in the experiment section.

4.2.2 *Representing Anchor Concept.*

Each anchor concept corresponds to one node in the existing taxonomy $\mathcal{T}^0$ that could be the "*parent*" of a query concept. One naïve way to represent an anchor concept is to directly use its initial feature vector. A key limitation of this approach is that it captures only the "*parent*" node information and loses other surrounding nodes' signals. We illustrate this limitation below:

Example 2. *Suppose we are given a query concept* "high dependency unit" *to predict whether it should be under the* "hospital room" *node in an existing taxonomy. As these two concepts have dissimilar embeddings based on their surface names, we may believe this query concept shouldn't be placed underneath this anchor concept. However, if we know that this anchor concept has two children nodes,* i.e., "intensive care unit" *and* "low dependency unit"*, that are closely related to the query concept, we are more likely to put the query concept under this anchor concept, correctly.*

The above example demonstrates the importance of capturing local structure information in the anchor concept representation. We model the anchor concept using its ego network. Specifically, we consider the anchor concept to be the "*parent*" node of a query concept. The ego network of the anchor concept consists of the "*sibling*" nodes and "*grand parent*" nodes of the query concept, as shown in Figure 2. We represent the anchor concept based on its ego network using a graph neural network.

Graph Neural Network Architectures. Given an anchor concept a_i with its corresponding ego network G_{a_i} and its initial representation a_i, we use a graph neural network (GNN) to generate its final representation $\mathbf{a}_i$. This GNN contains two components: (1) a *graph propagation* module that transforms and propagates node features over the graph structure to compute individual node embeddings in G_{a_i}, and (2) a *graph readout* module that combines node embeddings into the full ego network embedding which encodes all local structure information centered around the anchor concept.

A graph propagation module uses a neighborhood aggregation strategy to iteratively update the representation of a node u by aggregating representations of its neighbors $N(u)$ and itself. We denote $N(u) \cup \{u\}$ as $\widetilde{N(u)}$. After K iterations, a node's representation captures the structural information within its K-hop neighborhood. Formally, we define a GNN with K-layers as follows:

$$h_u^{(k)} = \text{AGG}^{(k)}\left(\{h_v^{(k-1)} | v \in \widetilde{N(u)}\}\right), \quad k \in \{1, \ldots, K\}, \tag{2}$$

where $h_u^{(k)}$ is node u's feature in the k-th layer; $h_u^{(0)}$ is node u's initial feature vector, and $\text{AGG}^{(k)}$ is an aggregation function in the k-th layer. We instantiate $\text{AGG}^{(k)}$ using two popular architectures: Graph Convolutional Network (GCN) [19] and Graph Attention Network (GAT) [43]. GCN defines the AGG function as follows:

$$\text{AGG}^{(k)}\left(\{h_v^{(k-1)} | v \in \widetilde{N(u)}\}\right) = \rho\left(\sum_{v \in \widetilde{N(u)}} \alpha_{uv}^{(k-1)} \mathbf{W}^{(k-1)} h_v^{(k-1)}\right), \tag{3}$$

where $\alpha_{uv}^{(k-1)} = 1/\sqrt{|\widetilde{N(u)}||\widetilde{N(v)}|}$ is a normalization constant (same for all layers); ρ is a non-linear function (*e.g.*, ReLU), and $\mathbf{W}^{(k-1)}$ is the learnable weight matrix. If we interpret $\alpha_{uv}^{(k-1)}$ as the *importance* of node v's feature to node u, GCN calculates it using only the graph structure without leveraging the node features. GAT addresses this limitation by defining $\alpha_{uv}^{(k-1)}$ as follows:

$$\alpha_{uv}^{(k-1)} = \frac{\exp\left(\gamma\left(\mathbf{z}^{(k-1)}[\mathbf{W}^{(k-1)} h_u^{(k-1)} \| \mathbf{W}^{(k-1)} h_v^{(k-1)}]\right)\right)}{\sum_{v' \in \widetilde{N(u)}} \exp\left(\gamma\left(\mathbf{z}^{(k-1)}[\mathbf{W}^{(k-1)} h_u^{(k-1)} \| \mathbf{W}^{(k-1)} h_{v'}^{(k-1)}]\right)\right)}, \tag{4}$$

where both $\mathbf{z}^{(k-1)}$ and $\mathbf{W}^{(k-1)}$ are learnable parameters; $\gamma(\cdot)$ is another non-linear function (*e.g.*, LeakyReLU), and "$\|$" represents the concatenation operation. Plugging the above $\alpha_{uv}^{(k-1)}$ into Eq. (3) we obtain the aggregation function in a *single-head* GAT. Finally, We execute M independent transformations of Eq. (3) and concatenate

their output features to compose the final output embedding of node u. This defines the aggregation function in a *multi-head* GAT (with M heads) as follows:

$$\text{AGG}^{(k)}\left(\{h_v^{(k-1)}|v \in \widetilde{N(u)}\}\right) = \Big\|_{m=1}^{M} \rho\left(\sum_{v \in \widetilde{N(u)}} \alpha_{uv}^{(k-1)} \mathbf{W}_m^{(k-1)} h_v^{(k-1)}\right), \tag{5}$$

where $\mathbf{W}_m^{(k-1)}$ is the m-th weight matrix in the m-th attention head.

After obtaining each node's final representation $h_u^{(K)}$, we generate the ego network's representation h_G using a graph readout module as follows:

$$h_G = \text{READOUT}(\{h_u^{(K)}|u \in G\}), \tag{6}$$

where READOUT is a permutation invariant function [51] such as element-wise mean or sum.

Position-enhanced Graph Neural Networks. One key limitation of the above GNN model is that they fail to capture each node's position information relative to the query concept. Take Figure 2 as an example, the "*hospital room*" node in the left ego network is the anchor node itself while in the right ego network it is the child of the anchor node. Such position information will influence how node feature propagates within the ego network and how the final graph embedding is aggregated.

An important innovation in TaxoExpan is the design of position-enhanced graph neural networks. The key idea is to learn a set of "position embeddings" and enrich each node feature with its corresponding position embedding. We denote node u's position as p_u and its position embedding at k-th layer as $\mathbf{p}_u^{(k)}$. We replace each node feature $h_u^{(k-1)}$ with its position-enhanced version $h_u^{(k-1)} \| \mathbf{p}_u^{(k-1)}$ in Eqs. (3-5) and adjust the dimensionality of $\mathbf{W}^{(k-1)}$ accordingly. Such position embeddings help us to learn better node representations from two aspects. First, we can capture more neighborhood information. Take $\mathbf{W}^{(k-1)} h_v^{(k-1)}$ in the right hand side of Eq. (3) as an example, we enhance it to the following:

$$\left[\mathbf{W}^{(k-1)} \| \mathbf{O}^{(k-1)}\right] \left[h_v^{(k-1)} \| \mathbf{p}_v^{(k-1)}\right] = \mathbf{W}^{(k-1)} h_v^{(k-1)} + \mathbf{O}^{(k-1)} \mathbf{p}_v^{(k-1)},$$

where $\mathbf{O}^{(k-1)}$ is another weight matrix used to transform position embeddings. The above equation shows that a node's new representation is jointly determined by its neighborhoods' contents (*i.e.*, $h_v^{(k-1)}$) and relative positions in the ego network (*i.e.*, $\mathbf{p}_v^{(k-1)}$). Second, for GAT architecture, we can better model neighbor importance as the term $\alpha_{uv}^{(k-1)}$ in Eq. (3) currently depends on both $\mathbf{p}_u^{(k-1)}$ and $\mathbf{p}_v^{(k-1)}$.

Furthermore, we propose two schemes to inject position information in the graph readout module. The first one, called weighted mean readout (WMR), is defined as follows:

$$\text{READOUT}(\{h_u^{(K)}|u \in G\}) = \sum_{u \in G} \frac{\log(1+\exp(\alpha_{p_u}))}{\sum_{u' \in G} \log(1+\exp(\alpha_{p'_u}))} h_u^{(K)}, \tag{7}$$

where α_{p_u} is the parameter indicating the importance of position p_u. The second scheme is called concatenation readout (CR) which combines the average embeddings of nodes with the same position as follows:

$$\text{READOUT}(\{h_u^{(K)}|u \in G\}) = \Big\|_{p \in \mathcal{P}} \frac{\mathcal{I}(p_u = p) h_u^{(K)}}{\sum_{u' \in G} \mathcal{I}(p_{u'} = p)}, \tag{8}$$

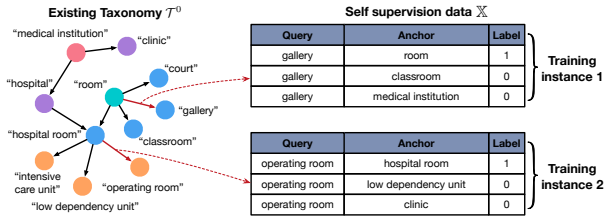

Figure 4: Self-supervision generation.

where $\mathcal{P}$ is the set of all positions we are modeling and $\mathcal{I}(\cdot)$ is an indicator function which returns 1 if its internal statement is true and returns 0 otherwise.

4.2.3 *Matching Query Concept and Anchor Concept.*

Based on the learned query concept representation $\mathbf{n}_i \in \mathbb{R}^{D_1}$ and anchor concept representation $\mathbf{a}_i \in \mathbb{R}^{D_2}$, we calculate their match score using a matching module $f(\cdot) : \mathbb{R}^{D_2} \times \mathbb{R}^{D_1} \to \mathbb{R}$. We study two architectures. The first one is a multi-layer perceptron with one hidden layer, defined as follows:

$$f^{\text{MLP}}(\mathbf{a}_i, \mathbf{n}_i) = \sigma\left(\mathbf{W}_2 \gamma(\mathbf{W}_1(\mathbf{a}_i \| \mathbf{n}_i) + \mathbf{B}_1) + \mathbf{B}_2\right), \tag{9}$$

where $\{\mathbf{W}_1, \mathbf{B}_1, \mathbf{W}_2, \mathbf{B}_2\}$ are parameters; $\sigma(\cdot)$ is the sigmoid function, and $\gamma(\cdot)$ is the LeakyReLU activation function. The second architecture is a log-bilinear model defined as follows:

$$f^{\text{LBM}}(\mathbf{a}_i, \mathbf{n}_i) = \exp\left(\mathbf{a}_i^T \mathbf{W} \mathbf{n}_i\right), \tag{10}$$

where $\mathbf{W}$ is a learnable interaction matrix. We choose these MLP and LBM as they are representative architecures in linear and bilinear interaction models, respectively.

4.3 Model Learning and Inference

The above section discusses how to model query-anchor matching using a parameterized function $f(\cdot|\Theta)$. In this section, we first introduce how we learn those parameters Θ using self-supervision from the existing taxonomy. Then, we establish the connection between the matching score with the conditional probability $\mathbf{P}(n_i|a_i)$, and discuss how to conduct model inference.

Self-supervision Generation. Figure 4 shows the generation process of self supervision data. Given one edge $\langle n_p, n_c \rangle$ in the existing taxonomy $\mathcal{T}^0 = (\mathcal{N}^0, \mathcal{E}^0)$, we first construct a positive $\langle$anchor, query$\rangle$ pair by using child node n_c as the "*query*" and parent node n_p as the "*anchor*". Then, we construct N negative pairs by fixing the query node n_c and randomly selecting N nodes $\{n_r^l|_{l=1}^N\} \subset \mathcal{N}^0$ that are neither parents nor descendants of n_c. These $N+1$ pairs (one positive and N negatives) collectively consist of one training instance $\mathbf{X} = \{\langle n_p, n_c \rangle, \langle n_r^1, n_c \rangle, \ldots, \langle n_r^N, n_c \rangle\}$. By repeating the above process for each edge in $\mathcal{T}^0$, we obtain the full self-supervision dataset $\mathbb{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_{|\mathcal{E}^0|}\}$. Notice that a node with C parents in $\mathcal{T}^0$ will derive C training instances in $\mathbb{X}$.

Model Training. We learn our model on $\mathbb{X}$ using the InfoNCE loss [41] as follows:

$$\mathcal{L}(\Theta) = -\frac{1}{|\mathbb{X}|} \sum_{\mathbf{X}_i \in \mathbb{X}} \left[\log \frac{f(n_p, n_c)}{\sum_{\langle n_j, n_c \rangle \in \mathbf{X}_i} f(n_j, n_c)}\right], \tag{11}$$

where the subscript $j \in [1, 2, \ldots, N+1]$. If $j = 1$, $\langle n_j, n_c \rangle$ is a positive pair, otherwise, $\langle n_j, n_c \rangle$ is a negative pair. The above loss is

the cross entropy of classifying the positive pair $\langle n_p, n_c \rangle$ correctly, with $\frac{f(n_p, n_c)}{\sum_{\langle n_j, n_c \rangle \in \mathbb{X}_i} f(n_j, n_c)}$ as the model prediction. Optimizing this loss results in $f(a_i, n_i)$ estimating the following probability density (up to a multiplicative constant):

$$f(a_i, n_i) \propto \frac{\mathbf{P}(a_i|n_i)}{\mathbf{P}(a_i)}. \tag{12}$$

We prove the above result in Appendix and summarize our self-learning procedure in Algorithm 1. We establish the connection between matching score $f(a_i, n_i)$ with the probability $\mathbf{P}(n_i|a_i)$ in Eq. 1 as follows:

$$\mathbf{P}(n_i|a_i) = \frac{\mathbf{P}(a_i|n_i)}{\mathbf{P}(a_i)} \cdot \mathbf{P}(n_i) \propto f(a_i, n_i) \cdot \mathbf{P}(n_i). \tag{13}$$

We elaborate the implication of this equation below.

Model Inference. At the inference stage, we are given a new query concept n_i and apply the learned model $f(\cdot|\Theta)$ to predict its parent node in the existing taxonomy $\mathcal{T}^0$. Mathematically, we aim to find the anchor position a_i that maximizes $\mathbf{P}(n_i|a_i)$, which is equivalent to maximizing $f(a_i, n_i)$ because of Eq. (13) and the fact that $P(n_i)$ is the same across all positions. Therefore, we rank all candidate positions a_i based on their matching scores with n_i and select the top ranked one as the predicted parent node of this query concept. Although we currently select only the top one as query's single parent, we can also choose top-k ones as query's parents, if needed.

Summary. Given an existing taxonomy and a set of new concepts, our TaxoExpan first generates a set of self-supervision data and learns its internal model parameters using Algorithm 1. For each new concept, we run the inference procedure and find its best parent node in the existing taxonomy. Finally, we place these new concepts underneath their predicted parents one at a time, and output the expanded taxonomy.

Computational Complexity Analysis. At the training stage, our model uses $|\mathcal{E}^{(0)}|$ training instances every epoch and thus scales linearly to the number of edges in the existing taxonomy. At the inference stage, for each query concept, we calculate $|\mathcal{N}^{(0)}|$ matching scores, one for every existing node in $\mathcal{T}^0$. Although such $O(|\mathcal{N}^{(0)}|)$ cost per query is expensive, we can significantly reduce it using two strategies. First, most computation efforts of TaxoExpan are matrix multiplications and thus we use GPU for acceleration. Second, as the graph propagation and graph readout modules are query-independent (c.f. Fig. 4), we pre-compute all anchor representations and cache them. When a set of queries are given, we only run the matching module.

5 EXPERIMENTS

In this section, we study the performance of TaxoExpan on three large-scale real-world taxonomies.

5.1 Expanding MAG Field-of-Study Taxonomy

5.1.1 Datasets. We evaluate TaxoExpan on the public Field-of-Study (FoS) Taxonomy[2] in Microsoft Academic Graph (MAG) [38]. This FoS taxonomy contains over 660 thousand scientific concepts and more than 700 thousand taxonomic relations. Although being constructed semi-automatically, this taxonomy is of high quality,

[2]https://docs.microsoft.com/en-us/academic-services/graph/reference-data-schema

Algorithm 1: Self-supervised learning of TaxoExpan

Input: A taxonomy $\mathcal{T}^0$; negative size N, batch size B; model $f(\cdot|\Theta)$.
Output: Learned model parameters Θ.

```
Randomly initialize Θ;
while L(Θ) in Eq. (11) not converge do
    Enumerate edges in T^0 and sample B edges without replacement;
    X = { } # current batch of training instances;
    for each sampled edge ⟨n_p, n_c⟩ do
        Generate N negative pairs {⟨n_r^l, n_c⟩|_{l=1}^N};
        X ← X ∪ {⟨n_p, n_c⟩, ⟨n_r^1, n_c⟩, ..., ⟨n_r^N, n_c⟩};
    Update Θ based on X.
Return Θ;
```

Table 1: Dataset Statistics. $|\mathcal{N}|$ and $|\mathcal{E}|$ are the number of nodes and edges in the existing taxonomy. $|\mathcal{D}|$ indicates the taxonomy depth and $|C|$ is the number of new concepts.

Dataset	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{D}\rvert$	$\lvert C\rvert$
MAG-CS	24,754	42,329	6	2,450
MAG-Full	355,808	638,674	6	37,804
SemEval	95,882	89,089	20	600

as shown in the previous study [35]. Thus we treat each concept's original parent nodes as its correct anchor positions. We remove all concepts that have no relation in the original FoS taxonomy and then randomly mask 20% of leaf concepts (along with their relations) for validation and testing[3]. The remaining FoS taxonomy is then treated as the input existing taxonomy. We refer to this dataset as **MAG-Full**. Based on MAG-Full, we construct another dataset focusing on the computer science domain. Specifically, we first select a subgraph consisting of all descendants of "computer science" node and then mask 10% of leaf concepts in this subgraph for validation and another 10% of leaf nodes for testing. We name this dataset as **MAG-CS**.

To obtain the initial feature vector, we first construct a corpus that consists of all paper abstracts mentioning at least one concept in the original MAG dataset. Then, we use "_" to concatenate all tokens in one concept (*e.g.*, "machine learning" → "machine_learning") and learn 250-dimension word embeddings using skipgram model in word2vec[4] [28]. Finally, we use these learned embeddings as the initial feature vector. Table 1 lists the statistics of these two datasets. All datasets and our model implementations are available at: https://github.com/mickeystroller/TaxoExpan.

5.1.2 Evaluation Metrics. As our model returns a rank list of all candidate parents for each input query concept, we evaluate its performance using the following three ranking-based metrics.

- **Mean Rank (MR)** measures the average rank position of a query concept's true parent among all candidates. For queries with multiple parents, we first calculate the rank position of each individual parent and then take the average of all rank positions. Smaller MR value indicates better model performance.

[3]Here we mask only leaves because if we remove intermediate nodes, we have to remove their descendants from the candidate parent pool, which causes different masked nodes (as testing query concepts) having different candidate pools.

[4]We also test CBOW model, fastText [5] and BERT embedding [8] (averaged across all concept mentions), and empirically we find skipgram model in word2vec works best on this dataset.

Table 2: Overall results on MAG-CS and MAG-Full datasets. We run all methods three times and report the averaged result with standard deviation. Note that smaller MR indicates better model performance. For all other metrics, larger values indicate better performance. We highlight the best two models in terms of the average performance under each metric.

Method	MAG-CS				MAG-Full			
	MR	Hit@1	Hit@3	MRR	MR	Hit@1	Hit@3	MRR
Closest-Parent	1327.16 (±0.000)	0.0531 (±0.000)	0.0986 (±0.000)	0.2691 (±0.000)	14355.5 (±0.000)	0.0360 (±0.000)	0.0728 (±0.000)	0.1897 (±0.000)
Closest-Neighbor	382.07 (±0.000)	0.1085 (±0.000)	0.2000 (±0.000)	0.3987 (±0.000)	4160.8 (±0.000)	0.0221 (±0.000)	0.0419 (±0.000)	0.1405 (±0.000)
dist-XGBoost	136.86 (±1.832)	0.1903 (±0.010)	0.3483 (±0.014)	0.6618 (±0.003)	**426.70 (±8.047)**	**0.1498 (±0.076)**	**0.3046 (±0.009)**	0.5621 (±0.002)
ParentMLP	**114.79 (±12.25)**	0.0729 (±0.088)	0.2656 (±0.037)	0.6454 (±0.009)	457.14 (±39.81)	0.098 (±0.094)	0.1928 (±0.086)	0.4950 (±0.012)
DeepSetMLP	115.26 (±9.159)	**0.1988 (±0.005)**	**0.3581 (±0.016)**	**0.6653 (±0.015)**	444.83 (±27.59)	0.1461 (±0.005)	0.2971 (±0.064)	**0.6392 (±0.017)**
TaxoExpan	**80.33 (±5.470)**	**0.2121 (±0.010)**	**0.3823 (±0.012)**	**0.6929 (±0.003)**	**341.31 (±33.62)**	**0.1523 (±0.009)**	**0.3087 (±0.010)**	**0.6453 (±0.035)**

- **Hit@k** is the number of query concepts whose parent is ranked in the top k positions, divided by the total number of queries.
- **Mean Reciprocal Rank (MRR)** calculates the reciprocal rank of a query concept's true parent. We follow [47] and use a scaled version of MRR in the below equation:

$$\text{MRR} = \frac{1}{|C|} \sum_{c \in C} \frac{1}{|parent(c)|} \sum_{i \in parent(c)} \frac{1}{\lceil R_{i,c}/10 \rceil},$$

where $parent(c)$ represents the parent node set of the query concept c, and $R_{i,c}$ is the rank position of query concept c's true parent i. We scale the original MRR by a factor 10 in order to amplify the performance gap between different methods.

5.1.3 Compared Methods. We compare the following methods:

(1) **Closest-Parent**: A rule-based method which first scores each candidate position in the existing taxonomy based on its cosine distance to the query concept between their initial embedding, and then ranks all positions using this score. The position with the smallest distance is chosen to be query concept's parent.
(2) **Closest-Neighbor**: Another rule-based method that scores each position based on its distance to the query concept plus the average distance between its children nodes and the query.
(3) **dist-XGBoost**: A self-supervised boosting method that works directly on 39 manually-designed features generated using initial node embeddings without any embedding transformation. We input these features into XGBoost [6], a tree-based boosting model, to predict the matching score between a query concept and a candidate position.
(4) **ParentMLP**: A self-supervised method that first concatenates the query concept embedding with the candidate position embedding and then feeds them into a Multi-Layer Perceptron (MLP) for prediction.
(5) **DeepSetMLP**: Another self-supervised method that extends ParentMLP by adding information of candidate position's children nodes. Specifically, we first use DeepSet architecture [51] to generate the representation of the children node set and then concatenate it with query & candidate position representations before the final MLP module.
(6) **TaxoExpan**: Our proposed framework using position-enhanced GAT (PGAT) as graph propagation module, weighted mean readout (WMR) for graph readout, and log-bilinear model (LBM) for query-anchor matching. We learn this model using our proposed InfoNCE loss.

5.1.4 Implementation Details and Parameter Settings. For a fair comparison, we use the same 250-dimension embeddings across all compared methods. We use Google's original word2vec implementation[5] for learning embeddings and employ gensim[6] to load trained embeddings for calculating term distances in Closest-Parent, Closest-Neighbor, and dist-XGBoost methods. For the other three methods, we implement them using PyTorch and DGL framework[7]. We tune hyper-parameters in all self-supervised methods on the masked validation set. For TaxoExpan, we use a two-layer position-enhanced GAT where the first layer has four attention heads (of size 250) and the second layer has one attention head (of size 500). For both layers, we use 50-dimension position embeddings and apply dropout with rate 0.1 on the input feature vectors. We use Adam optimizer with initial learning rate 0.001 and ReduceLROnPlateau scheduler[8] with three patience epochs. We discuss the influence of these hyper-parameters in the next subsection.

Table 3: Ablation analysis of model architectures on MAG-CS dataset. We assign an index to each model variant (shown in the first column). All models are run three times with their averaged scores reported.

Ind	Graph Propagate	Graph Readout	Matching	MR	Hit@1	Hit@3	MRR
1	GCN	Mean	MLP	167.82	0.1581	0.2964	0.6002
2	GAT	Mean	MLP	131.46	0.1584	0.3192	0.6409
3	PGCN	Mean	MLP	148.54	0.1809	0.3015	0.6255
4	**PGAT**	**Mean**	**MLP**	**100.80**	**0.1896**	**0.3304**	**0.6525**
5	PGCN	WMR	MLP	144.81	0.1798	0.3014	0.6309
6	PGCN	CR	MLP	135.89	0.1902	0.3118	0.6348
7	**PGAT**	**WMR**	**MLP**	**92.62**	**0.1945**	**0.3584**	**0.6619**
8	PGAT	CR	MLP	95.84	0.1897	0.3512	0.6596
9	PGCN	WMR	LBM	139.41	0.1829	0.3370	0.6642
10	PGCN	CR	LBM	130.12	0.1934	0.3462	0.6776
11	**PGAT**	**WMR**	**LBM**	**80.33**	**0.2121**	**0.3823**	**0.6929**
12	PGAT	CR	LBM	84.40	0.2089	0.3813	0.6894

5.1.5 Experimental Results. We present the experimental results in the following aspects.

1. Overall Performance. Table 2 presents the results of all compared methods. First, we find that Closest-Neighbor method clearly outperforms Closest-Parent method and DeepSetMLP is much better than ParentMLP. This demonstrates the effectiveness of modeling local structure information. Second, we compare dist-XGBoost method with Closest-Neighbor and show that self-supervision indeed helps us to learn an effective way to combine various neighbor distance information. All four self-supervised methods outperform

[5] https://github.com/tmikolov/word2vec
[6] https://github.com/RaRe-Technologies/gensim
[7] https://github.com/dmlc/dgl
[8] https://pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.ReduceLROnPlateau

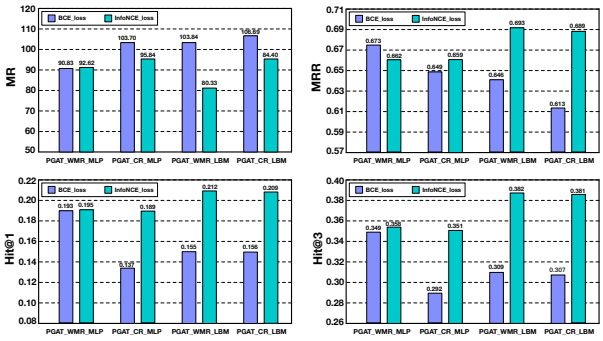

Figure 5: Ablation analysis of training schemes on MAG-CS dataset. We compare models trained using Binary Cross Entropy (BCE) loss with those trained using InfoNCE loss.

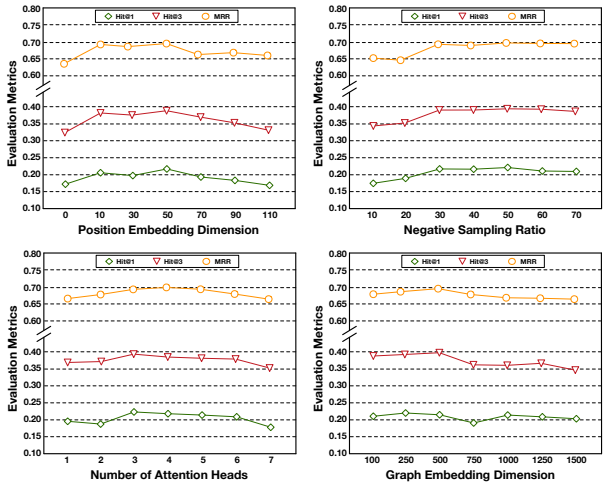

Figure 6: Hyper-parameter sensitivity analysis on MAG-CS dataset. We use PGAT for graph propagation, WMR for graph readout, and LBM for query-graph matching. Model is trained using InfoNCE loss.

rule-based methods. Finally, our proposed TaxoExpan has the overall best performance across all the metrics and defeats the second best method by a large margin.

2. Ablation Analysis of Model Architectures. TaxoExpan contains three key components: a graph propagation module, a graph readout module, and a matching model. Here, we study how different choices of these components affect the performance of TaxoExpan. Table 3 lists the results and the first column contains the index of each model invariant.

First, we analyze graph propagation module by using simple average scheme for graph readout and MLP for matching. By comparing model 1 to model 3 and model 2 to model 4, we can see that graph attention architecture (GAT) is better than graph convolution architecture (GCN). Furthermore, the position-enhanced variants clearly outperform their non-position counterparts (model 3 versus model 1 and model 4 versus model 2). This illustrates the efficacy of the position embeddings in the graph propagation module.

Second, we study graph readout module by fixing the graph propagation module to be the best two variants among models 1-4. We can see both model 5 & 6 outperform model 3 and model 7 & 8 outperform model 4. This signifies that the position information also helps in the graph readout module. However, the best strategy of incorporating position information depends on the graph propagation module. The concatenation readout scheme works better for PGCN while the weighted mean readout is better for PGAT. One possible explanation is that the concatenation readout leads to more parameters in matching model and as PGAT itself has more parameters than PGCN, further introducing more parameters in PGAT may cause the model to be overfitted.

Finally, we examine the effectiveness of different matching models. We replace the MLP in models 5-8 with LBM to create model variants 9-12. We can clearly see that LBM works better than MLP. It could be that LBM better captures the interaction between the query representation and the final anchor representation.

3. Ablation Analysis of Training Schemes. In this subsection, we evaluate the effectiveness of our proposed training scheme. In this study, we first group a set of positive and negative $\langle query, anchor \rangle$ pairs into *one single* training instance (c.f. Sect. 4.3) and learn the model using InfoNCE loss (c.f. Eq. (11)). An alternative is to treat these pairs as different instances and train the model using standard binary cross entropy (BCE) loss. Under this training scheme, we formulate our problem as a binary classification task. We compare these two training schemes for the top 4 best models in Table 3 (*i.e.*, model 7, 8, 11, and 12). Results are shown in Figure 5. Our proposed training scheme with InfoNCE loss is overall much better, it beats the BCE loss scheme on 14 out of total 16 cases. One reason is that BCE loss is very sensitive to the noises in the generated self-supervision data while InfoNCE loss is more robust to such label noise. Furthermore, we find that LBM matching can benefit more from our training scheme with InfoNCE loss – with larger margin on all 8 cases, compared with the simple MLP matching.

4. Hyper-parameter Sensitivity Analysis. We analyze how some hyper-parameters in TaxoExpan affect the performance in Figure 6. First, we find that choosing an approximate position embedding dimension is important. The model performance increases as this dimensionality increases until it reaches about 50. When we further increase position embedding dimension, the model will overfit and the performance decreases. Second, we study the effect of negative sampling ratio N. As shown in Figure 6, the model performance first increases as N increases until it reaches about 30 and then becomes stable. Finally, we examine two hyper-parameters controlling the model complexity: the number of heads in PGAT and the final graph embedding dimension. We observe that the best model performance is reached when the number of attention heads falls in range 3 to 5 and the graph embedding dimension is set to 500. Too many attention heads or too large graph embedding dimension will lead to overfit and performance degradation.

5. Efficiency and Scalability. We further analyze the scalability of TaxoExpan and its efficiency during model inference stage. Figure 7 (left) tests the model scalability by running on MAG-CS dataset sampled using different ratios. The training time (of 20 epochs) are measured on one single K80 GPU. TaxoExpan demonstrates a linear runtime trend, which validates our complexity analysis in Sect. 4.3. Second, Figure 7 (right) shows that TaxoExpan is very

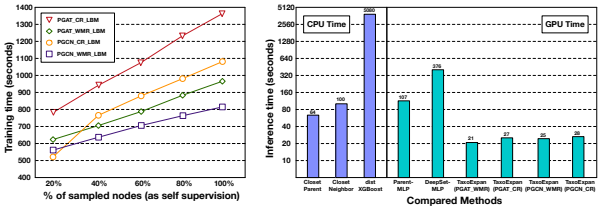

Figure 7: (Left) Training time of 20 epochs on GPU with respect to % of sampled nodes in the existing taxonomy. (Right) Inference time of all 2450 queries in MAG-CS dataset. Note here y-axis is in logarithm scale.

efficient during model inference stage. Using GPU, TaxoExpan takes less than 30 seconds to predict the anchor positions for all 2450 new query concepts.

6. Case Study. Figure 8 shows some outputs of TaxoExpan on both MAG-CS and MAG-Full datasets. On MAG-CS dataset, we can see that over 20% of queries have their true parents correctly ranked at the first position and less than 1.5% queries have their "true" parents ranked outside of top 1000 positions. Among these 1.5% significantly wrong queries, we find some of them actually have incorrect existing parents. For example, the concept "*boils and carbuncles*", which is a disease entity, is mistakenly put under parent node "*dataset*". Similar cases also happen on MAG-Full dataset where we find the concept "*blood staining*" is currently under "*laryngeal mask airway*".

Besides the above label errors, we also observe two common mistake patterns. The first type of mistakes is caused by term ambiguity. For instance, the term "*java*" in concept "*java apple*" refers to an island in Indonesia where fruit apple is produced, rather than a programming language used in Apple company. The second type of mistakes results from term granularity. For example, TaxoExpan outputs the two most likely parent nodes of concept "*captcha*" are "*artificial intelligence*" and "*computer security*". Although these two concepts are certainly relevant to "*captcha*", they are too general compared to its true parent node "*internet privacy*".

Finally, we observe that TaxoExpan can return very sensible anchor positions of query concepts, even though they are not exactly the current "true" parents. For example, the concept "*medline plus*" refers to a large online medical library and thus is related to both "*world wide web*" and "*library science*". Also, the concept "*email hacking*" is clearly relevant to both "*internet privacy*" and "*hacker*".

7. TaxoExpan for Taxonomy Self-Cleaning. From the above case studies, we find another interesting application of TaxoExpan is to use it for cleaning the existing taxonomy. Specifically, we partition all leaf nodes of the existing taxonomy into 5 groups and randomly mask one group of nodes. Then, we train a TaxoExpan model on the remaining nodes and predict on the masked leaf nodes. Next, we select those entities whose true parents appear at the bottom of the rank lists returned by TaxoExpan (*i.e.*, the long-tail part of two histograms in Figure 8). The parents of those selected entities are highly questionable and calls for further manual inspections. Our preliminary experiments on the MAG-CS taxonomy shows that about 30% of these entities have existing parent nodes which are less appropriate than the parents inferred by TaxoExpan.

5.2 Evaluation on SemEval Task Benchmark

5.2.1 Datasets. We further evaluate TaxoExpan using SemEval Task 14 Benchmark dataset[9] [18] which includes WordNet 3.0 as the existing taxonomy and additional 1,000 domain-specific concepts with manual labels, split into 400 training concepts and 600 testing concepts. Each concept is either a verb or a noun and has a textual definition of a few sentences. The original task goal is to enrich the taxonomy by performing two actions for each new concept: (1) *attach*, where a new concept is treated as a new synset and is attached as a hyponym of one existing synset in WordNet, and (2) *merge*, where a new concept is merged into an existing synset. However, previous state-of-the-art methods [18, 32, 42], including the winning solution, are only performing the *attach* operation. In this work, we also follow this convention and attach each new concept to the top-ranked synset in the WordNet. Finally, we obtain the initial feature vectors (for both new concepts and existing words in the WordNet) using pre-trained subword-aware fasttext embeddings[10]. For each concept, we generate its definition embedding and name embedding by averaging the embedding of each token in its textual definition and name string, correspondingly. Then, we sum the definition and name embeddings of a concept and use them as the initial embeddings for the TaxoExpan model.

5.2.2 Evaluation Metrics. We use the three official metrics defined in original SemEval Task 14 for evaluation:

(1) **Accuracy (Wu&P)** is the semantic similarity between a predicted parent node x_p and the true parent x_t, calculated as $\text{Wu\&P}(x_p, x_t) = \frac{2 \cdot depth_{LCA(x_p,x_t)}}{depth_{x_p} + depth_{x_t}}$, where $depth_x$ is the depth of node x is the WordNet taxonomy and $LCA(x_p, x_t)$ represents the Least Common Ancestor of x_p and x_t.
(2) **Recall** is the percentage of concepts for which an attached parent is predicted[11].
(3) **F1** is the harmonic mean of Wu&P accuracy and recall.

5.2.3 Baseline Methods. We compare the following methods:

(1) **FWFS** [18]: The original baseline in Task 14. Given a concept c with its definition d_c, this method picks the first word w in d_c that has the same part of speech as c and treats this word as the parent node of c.
(2) **MSejrKU** [32]: The winning solution of Task 14. This method leverages distributional and syntactic features to train a SVM classifier which is then used to predict the goodness of fit for a new concept with an existing synset in WordNet.
(3) **ETF** [42]: The current state-of-the-art method that learns a LambdaMART model with 15 manually designed features, including topological features from the taxonomy's graph structure and semantic features from corpus and Bing search results.
(4) **ETF-FWFS** [42]: The ensemble model of FWFS and ETF, which adds the FWFS property as a binary feature into the LambdaMART model in ETF.
(5) **dist-XGBoost**: The same tree boosting model described in the previous subsection 5.1.3.
(6) **TaxoExpan**: Our proposed taxonomy expansion framework.

[9] http://alt.qcri.org/semeval2016/task14/.
[10] We use the wiki-news-300d-1M-subword.vec.zip version on fastText official website.
[11] This metric is used because the original task allows a model to decline to place new concepts in order to avoid making placements with low confidence.

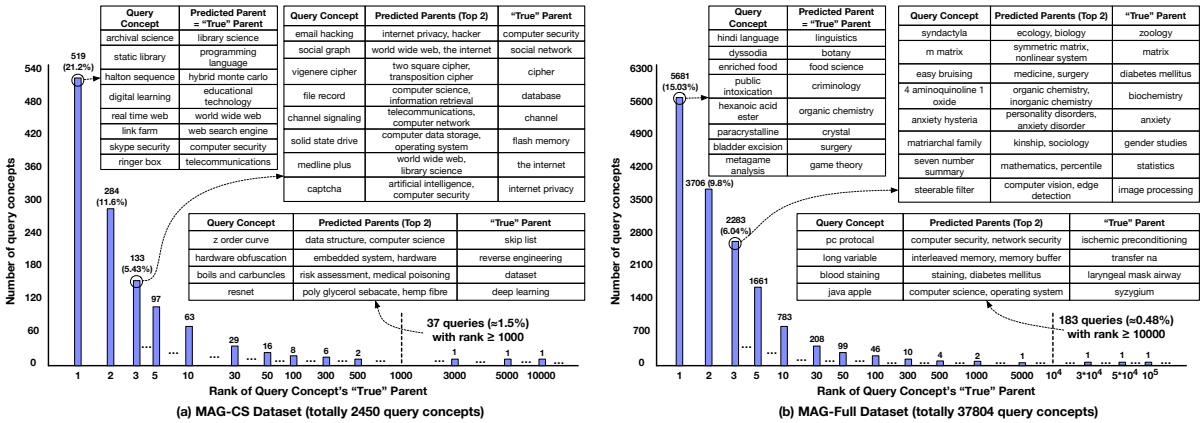

Figure 8: Example output of TaxoExpan on MAG-CS and MAG-Full datasets. We draw a histogram of the ranks of query concepts' true parents within the rank list returned by TaxoExpan. In subfigure (a), for example, we have 519 (out of 2450) queries that their parents are exactly ranked in the first position.

Table 4: Model performance on SemEval dataset. TaxoExpan versus all previous state-of-the-art methods. We report the best performance of all existing methods in the literature.

Method	Wu&P	Recall	F1
MSejrKU [32]	0.523	0.973	0.680
FWFS [18]	0.514	1.000	0.679
ETF [42]	0.473	1.000	0.642
ETF-FWFS [42]	0.562	1.000	0.720
dist-XGBoost	0.528	1.000	0.691
TaxoExpan	0.543	1.000	0.704
TaxoExpan-FWFS	0.566	1.000	0.723

(7) **TaxoExpan-FWFS**: Similar to ETF-FWFS, this is the ensemble model of FWFS and TaxoExpan. We treat the FWFS heuristic as a binary feature and add it into the final matching module.

For all previous methods, we directly report their best performance in the literature. For the remaining methods, we tune them following the same procedure described in the Section 5.1.4.

5.2.4 Experimental Results. Table 4 shows the experimental results on SemEval dataset. We can see that both dist-XGBoost and TaxoExpan methods can outperform the previous winning system of this task (*i.e.*, MSejrKU) and the baseline ETF. In addition, we can see the FWFS heuristic is indeed very powerful for this dataset and incorporating it as a strong feature can significantly boost the performance. Finally, we show that TaxoExpan-FWFS can achieve the new state-of-the-art performance on this dataset.

6 CONCLUSION

This paper studies taxonomy expansion when no human labeled supervision data are given. We propose a novel TaxoExpan framework which generates self-supervision data from the existing taxonomy and learns a position-enhanced GNN model for expansion. To make the best use of self-supervision data, we design a noise-robust objective for effective model training. Extensive experiments demonstrate the effectiveness and efficiency of TaxoExpan on three taxonomies from different domains. Interesting future work includes modeling inter-dependency among new concepts, leveraging current method to cleaning the input existing taxonomy, and incorporating feedbacks from downstream applications (*e.g.*, search & recommendation) to generate more diverse supervision signals for expanding the taxonomy.

7 ACKNOWLEDGEMENT

Research was sponsored in part by DARPA under Agreements No. W911NF-17-C-0099 and FA8750-19-2-1004, National Science Foundation IIS 16-18481, IIS 17-04532, and IIS-17-41317, and DTRA HDTRA11810026. Any opinions, findings, and conclusions or recommendations expressed in this document are those of the author(s) and should not be interpreted as the views of any U.S. Government.

APPENDIX

Proof of Loss Function

Here we prove that optimizing the loss function in Eq. (11) will result in $f(\cdot)$ estimating the probability density in Eq. (12). By construction, $\mathbf{X}$ contains query n_c's one positive anchor (*i.e.*, its true parent n_p) sampled from the true distribution $\mathbf{P}(a_i|n_c)$ and N negative anchors $\{n_r^l|_{l=1}^N\}$ sampled from a uniform distribution $\mathbf{P}(a_i)$. If we merge these $N+1$ anchors into a small set and consider the task of selecting true anchor n_p's position j^* in $[1, 2, \ldots, N+1]$, we can view Eq. (11) as the cross entropy of position distribution $\hat{\mathbf{P}}$ from model prediction relative to the true distribution $\mathbf{P}^*$. Specifically, the model predicted position distribution $\hat{\mathbf{P}}_j = \frac{f(a_j, n_c)}{\sum_{k=1}^{N+1} f(a_k, n_c)}$ where one of $\{a_k|_{k=1}^{N+1}\}$ is the true anchor and all the others are negative anchors. Meanwhile, in the true position distribution:

$$\mathbf{P}_j^* = \frac{\mathbf{P}(a_j|n_c)\prod_{l\neq j}\mathbf{P}(a_l)}{\sum_{k=1}^{N+1}\left(\mathbf{P}(a_k|n_c)\prod_{l\neq k}\mathbf{P}(a_l)\right)} = \frac{\frac{\mathbf{P}(a_j|n_c)}{\mathbf{P}(a_j)}}{\sum_{k=1}^{N+1}\frac{\mathbf{P}(a_k|n_c)}{\mathbf{P}(a_k)}}.$$

From above, we can see that the optimal value for $f(a_j, n_c)$ is proportional to $\frac{\mathbf{P}(a_j|n_c)}{\mathbf{P}(a_j)}$.

REFERENCES

[1] Rami Aly, Shantanu Acharya, Alexander Ossa, Arne Köhn, Christian Biemann, and Alexander Panchenko. 2019. Every Child Should Have Parents: A Taxonomy Refinement Algorithm Based on Hyperbolic Term Embeddings. In *ACL*.

[2] Luis Espinosa Anke, José Camacho-Collados, Claudio Delli Bovi, and Horacio Saggion. 2016. Supervised Distributional Hypernym Discovery via Domain Adaptation. In *EMNLP*.

[3] Luis Espinosa Anke, José Camacho-Collados, Sara Rodríguez-Fernández, Horacio Saggion, and Leo Wanner. 2016. Extending WordNet with Fine-Grained Collocational Information via Supervised Distributional Learning. In *COLING*.

[4] Mohit Bansal, David Burkett, Gerard de Melo, and Dan Klein. 2014. Structured Learning for Taxonomy Induction with Belief Propagation. In *ACL*.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606* (2016).

[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*.

[7] Anne Cocos, Marianna Apidianaki, and Chris Callison-Burch. 2018. Comparing Constraints for Taxonomic Organization. In *NAACL*.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[9] Christiane Fellbaum. 1998. *WordNet*.

[10] Christiane Fellbaum, Udo Hahn, and Barry D. Smith. 2006. Towards new information resources for public health - From WordNet to MedicalWordNet. *Journal of biomedical informatics* (2006).

[11] Amit Gupta, Rémi Lebret, Hamza Harkous, and Karl Aberer. 2017. Taxonomy Induction Using Hypernym Subsequences. In *CIKM*.

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

[13] Marti A. Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. In *COLING*.

[14] Wen Hua, Zhongyuan Wang, Haixun Wang, Kai Zheng, and Xiaofang Zhou. 2017. Understand Short Texts by Harvesting and Analyzing Semantic Knowledge. *TKDE* (2017).

[15] Jun Huang, Zhaochun Ren, Wayne Xin Zhao, Gaole He, Ji-Rong Wen, and Daxiang Dong. 2019. Taxonomy-Aware Multi-Hop Reasoning Networks for Sequential Recommendation. In *WSDM*.

[16] Meng Jiang, Jingbo Shang, Taylor Cassidy, Xiang Ren, Lance M. Kaplan, Timothy P. Hanratty, and Jiawei Han. 2017. MetaPAD: Meta Pattern Discovery from Massive Text Corpora. In *KDD*.

[17] David Jurgens and Mohammad Taher Pilehvar. 2015. Reserating the awesometastic: An automatic extension of the WordNet taxonomy for novel terms. In *NAACL-HLT*.

[18] David Jurgens and Mohammad Taher Pilehvar. 2016. SemEval-2016 Task 14: Semantic Taxonomy Enrichment. In *SemEval@NAACL-HLT*.

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

[20] Zornitsa Kozareva and Eduard H. Hovy. 2010. A Semi-Supervised Method to Learn and Construct Taxonomies Using the Web. In *EMNLP*.

[21] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. 2018. Learning Deep Generative Models of Graphs. In *ICLR*.

[22] Dekang Lin. 1998. An Information-Theoretic Definition of Similarity. In *ICML*.

[23] Carolyn E. Lipscomb. 2000. Medical Subject Headings (MeSH). *Bulletin of the Medical Library Association* (2000).

[24] Bang Wu Liu, Weidong Guo, Di Niu, Chaoyue Wang, Shang-Zhong Xu, Jinghong Lin, Kunfeng Lai, and Yu Wei Xu. 2019. A User-Centered Concept Mining System for Query and Document Understanding at Tencent. In *KDD*.

[25] Anh Tuan Luu, Yi Tay, Siu Cheung Hui, and See-Kiong Ng. 2016. Learning Term Embeddings for Taxonomic Relation Identification Using Dynamic Weighting Neural Network. In *EMNLP*.

[26] Yuning Mao, Xiang Ren, Jiaming Shen, Xiaotao Gu, and Jiawei Han. 2018. End-to-End Reinforcement Learning for Automatic Taxonomy Induction. In *ACL*.

[27] Rui Meng, Yongxin Tong, Lei Chen, and Caleb Chen Cao. 2015. CrowdTC: Crowdsourced Taxonomy Construction. In *ICDM*.

[28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*.

[29] Ndapandula Nakashole, Gerhard Weikum, and Fabian M. Suchanek. 2012. PATTY: A Taxonomy of Relational Patterns with Semantic Types. In *EMNLP-CoNLL*.

[30] Vassilis Plachouras, Fabio Petroni, Timothy Nugent, and Jochen L. Leidner. 2018. A Comparison of Two Paraphrase Models for Taxonomy Augmentation. In *NAACL-HLT*.

[31] Stephen Roller, Douwe Kiela, and Maximilian Nickel. 2018. Hearst Patterns Revisited: Automatic Hypernym Detection from Large Text Corpora. In *ACL*.

[32] Michael Sejr Schlichtkrull and Héctor Martínez Alonso. 2016. MSejrKu at SemEval-2016 Task 14: Taxonomy Enrichment by Evidence Ranking. In *SemEval@NAACL-HLT*.

[33] Jiaming Shen, Zeqiu Wu, Dongming Lei, Jingbo Shang, Xiang Ren, and Jiawei Han. 2017. SetExpan: Corpus-Based Set Expansion via Context Feature Selection and Rank Ensemble. In *ECML/PKDD*.

[34] Jiaming Shen, Zeqiu Wu, Dongming Lei, Chao Zhang, Xiang Ren, Michelle T. Vanni, Brian M. Sadler, and Jiawei Han. 2018. HiExpan: Task-Guided Taxonomy Construction by Hierarchical Tree Expansion. In *KDD*.

[35] Zhihong Shen, Hao Ma, and Kuansan Wang. 2018. A Web-scale system for scientific knowledge exploration. In *ACL*.

[36] Yu Shi, Jiaming Shen, Yuchen Li, Naijing Zhang, Xinwei He, Zhengzhi Lou, Qi Zhu, Matthew D Walker, MyungâĂŘHwan Kim, and Jiawei Han. 2019. Discovering Hypernymy in Text-Rich Heterogeneous Information Network by Exploiting Context Granularity. In *CIKM'19*.

[37] Vered Shwartz, Yoav Goldberg, and Ido Dagan. 2016. Improving hypernymy detection with an integrated path-based and distributional method. *ACL* (2016).

[38] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In *WWW*.

[39] Darin Stewart. 2008. Building Enterprise Taxonomies.

[40] Antonio Toral, Rafael Muñoz, and Monica Monachini. 2008. Named Entity WordNet. In *LREC*.

[41] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *ArXiv* (2018).

[42] Nikhita Vedula, Patrick K. Nicholson, Deepak Ajwani, Sourav Dutta, Alessandra Sala, and Srinivasan Parthasarathy. 2018. Enriching Taxonomies With Functional Domain Knowledge. In *SIGIR*.

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

[44] Jingjing Wang, Changsung Kang, Yi Chang, and Jiawei Han. 2014. A hierarchical Dirichlet model for taxonomy expansion for search engines. In *WWW*.

[45] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q. Zhu. 2012. Probase: a probabilistic taxonomy for text understanding. In *SIGMOD Conference*.

[46] Grace Hui Yang. 2012. Constructing Task-Specific Taxonomies for Document Collection Browsing. In *EMNLP-CoNLL*.

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

[48] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*.

[49] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, and Jure Leskovec. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *NeurIPS*.

[50] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware Graph Neural Networks. In *ICML*.

[51] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. In *NIPS*.

[52] Chao Zhang, Fangbo Tao, Xiusi Chen, Jiaming Shen, Meng Jiang, Brian M. Sadler, Michelle T. Vanni, and Jiawei Han. 2018. TaxoGen: Constructing Topical Concept Taxonomy by Adaptive Term Embedding and Clustering. In *KDD*.

[53] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI*.

[54] Xiangling Zhang, Yueguo Chen, Jun Chen, Xiaoyong Du, Ke Wang, and Ji-Rong Wen. 2017. Entity Set Expansion via Knowledge Graphs. In *SIGIR '17*.

[55] Yuchen Zhang, Amr Ahmed, Vanja Josifovski, and Alexander J. Smola. 2014. Taxonomy discovery for personalized recommendation. In *WSDM*.