STEAM: Self-Supervised Taxonomy Expansion with Mini-Paths

Yue Yu
Georgia Institute of Technology
Atlanta, GA, USA
yueyu@gatech.edu

Yinghao Li
Georgia Institute of Technology
Atlanta, GA, USA
yinghaoli@gatech.edu

Jiaming Shen
University of Illinois at Urbana-Champaign
Urbana, IL, USA
js2@illinois.edu

Hao Feng
University of Electronic Science and Technology of China
Chengdu, Sichuan, China
is.fenghao@gmail.com

Jimeng Sun
University of Illinois at Urbana-Champaign
Urbana, IL, USA
jimeng@illinois.edu

Chao Zhang
Georgia Institute of Technology
Atlanta, GA, USA
chaozhang@gatech.edu

ABSTRACT

Taxonomies are important knowledge ontologies that underpin numerous applications on a daily basis, but many taxonomies used in practice suffer from the low coverage issue. We study the taxonomy expansion problem, which aims to expand existing taxonomies with new concept terms. We propose a self-supervised taxonomy expansion model named STEAM, which leverages natural supervision in the existing taxonomy for expansion. To generate natural self-supervision signals, STEAM samples mini-paths from the existing taxonomy, and formulates a node attachment prediction task between anchor mini-paths and query terms. To solve the node attachment task, it learns feature representations for query-anchor pairs from multiple views and performs multi-view co-training for prediction. Extensive experiments show that STEAM outperforms state-of-the-art methods for taxonomy expansion by 11.6% in accuracy and 7.0% in mean reciprocal rank on three public benchmarks. The code and data for STEAM can be found at https://github.com/yueyu1030/STEAM.

CCS CONCEPTS

• **Computing methodologies** → **Information extraction**.

KEYWORDS

Taxonomy Expansion, Mini-Paths, Self-supervised Learning

ACM Reference Format:
Yue Yu, Yinghao Li, Jiaming Shen, Hao Feng, Jimeng Sun, and Chao Zhang. 2020. STEAM: Self-Supervised Taxonomy Expansion with Mini-Paths. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403145

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

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403145

1 INTRODUCTION

Concept taxonomies play a central role in a wide spectrum of applications. On a daily basis, e-commerce websites like Amazon heavily rely on their product taxonomies to support billions of product navigations, searches [40], and recommendations [43]; scientific taxonomies (*e.g.*, MeSH[1]) make it much faster to identify relevant information from massive scientific papers, and concept taxonomies in knowledge bases (*e.g.*, Freebase [5]) underpin many question answering systems [14]. Due to such importance, many taxonomies have been curated in general and specific domains, *e.g.*, WordNet [25], Wikidata [37], MeSH [20], Amazon Product Taxonomy [16].

One bottleneck of many existing taxonomies is the *low coverage problem*. This problem arises mainly due to two reasons. First, many existing taxonomies are curated by domain experts. As the curation process is expensive and time-consuming, the result taxonomies often include only frequent and coarse-grained terms. Consequently, the curated taxonomies have high precision, but limited coverage. Second, domain-specific knowledge is constantly growing in most applications. New concepts arise continuously, but it is too tedious to rely on human curation to maintain and update the existing taxonomies. The low coverage issue can largely hurt the performance of downstream tasks, and automated taxonomy expansion methods are in urgent need.

Existing taxonomy construction methods follow two lines. One line is to construct taxonomies in an unsupervised way [21, 27, 38, 41]. This is achieved by hierarchical clustering [41], hierarchical topic modeling [21, 38], or syntactic patterns (*e.g.*, the Hearst pattern [15]). The other line adopts supervised approaches [13, 18, 23], which first detect hypernymy pairs (*i.e.*, term pairs with the "*is-a*" relation) and then organize these pairs into a tree structure. However, applying these methods for taxonomy *expansion* suffers from two limitations. First, most of them attempt to construct taxonomies *from scratch*. Their output taxonomies can rarely preserve the initial taxonomy structures curated by domain experts. Second, the performance of many methods relies on large amounts of annotated hypernymy pairs, which can be expensive to obtain in practice.

We propose a self-supervised taxonomy expansion model named STEAM[2], which leverages natural supervision in the existing taxonomy for expansion. To generate natural self-supervision signals,

[1] https://www.nlm.nih.gov/mesh/meshhome.html
[2] Short for **S**elf-supervised **T**axonomy **Exp**A**n**sion with **M**ini-Paths.

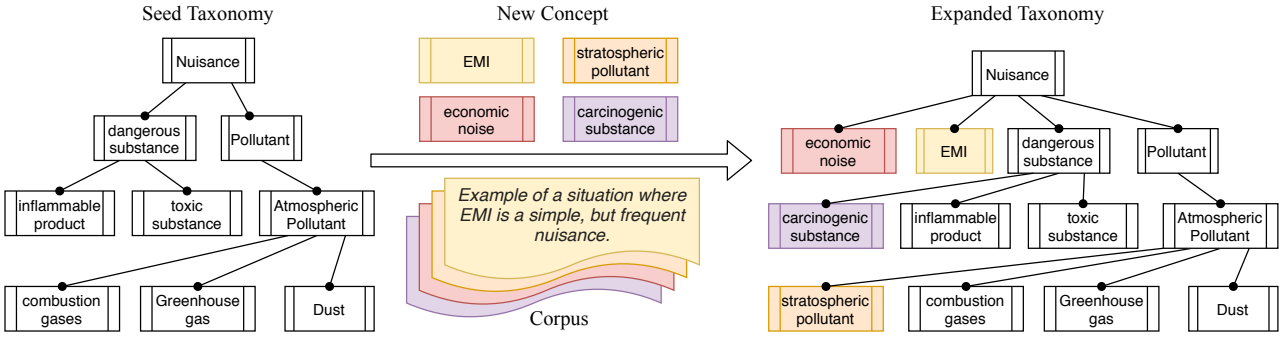


Figure 1: Illustration of the taxonomy expansion problem. Given an existing taxonomy, the task is to insert new concept terms (*e.g.*, *EMI, stratospheric pollutant, economic noise, carcinogenic substance*) into the correct positions in the existing taxonomy.

STEAM samples *mini-paths* from the existing taxonomy, and formulates a node attachment prediction task between mini-paths and query terms. The *mini-paths*, which contain terms in different layers (*e.g.* "*Pollutant*"–"*Atmospheric Pollutant*"–"*Dust*" in Figure 1), serve as candidate *anchors* for query terms and yield many training *query-anchor* pairs from the existing taxonomy. With these query-anchor pairs, we learn a model (Section 3.1) to pinpoint the correct position for a query term in the mini-path. Compared with previous methods [32, 35, 36] using single anchor terms, STEAM better leverages the existing taxonomy since the mini-paths contain richer structural information from different levels.

In cooperation with mini-path-based node attachment, STEAM extracts features for query-anchor pairs from multiple views, including: (1) *distributed features* that capture the similarity between terms' distributed representations; (2) *contextual features*, *i.e.* information from two terms' co-occurring sentences; (3) *lexico-syntactic features* extracted from the similarity of surface string names between terms. We find that different views can provide complementary information that is vital to taxonomy expansion. To fuse the three views more effectively, we propose a multi-view co-training procedure (Section 3.2). In this procedure, the three views lead to different branches for predicting the positions of the query term, and the predictions from these three views are encouraged to agree with each other.

We have conducted extensive experiments on three taxonomy construction benchmarks in different domains. The results show that STEAM outperforms state-of-the-art methods for taxonomy expansion by 11.6% in accuracy and 7.0% in mean reciprocal rank. Moreover, ablation studies demonstrate the effect of mini-path for capturing structural information from the taxonomy, as well as the multi-view co-training for harnessing the complementary signals from all views.

Our main contributions are: 1) a self-supervised framework that performs taxonomy expansion with natural supervision signals from existing taxonomies and text corpora; 2) a mini-path-based anchor format that better captures structural information in taxonomies for expansion; 3) a multi-view co-training procedure that integrates multiple sources of information in an end-to-end model; and 4) extensive experiments on several benchmarks verifying the efficacy of our method.

2 PROBLEM DESCRIPTION

We focus on the taxonomy expansion task for *term-level* taxonomies, which is formally defined as follows.

Definition 2.1 (Taxonomy). A taxonomy $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ is a tree structure where 1) $\mathcal{V}$ is a set of terms (words or phrases); and 2) $\mathcal{E}$ is a set of edges representing *is-a* relations between terms. Each directed edge $\langle v_i, v_j \rangle \in \mathcal{E}$ represents a hypernymy relation between term v_i and term v_j, where v_i is the hyponym (child) and v_j is the hypernym (parent).

The problem of taxonomy expansion (Figure 1) is to enrich an initial taxonomy by inserting new terms into it. These new terms are often automatically extracted and filtered from a text corpus. Formally, we define the problem as below:

Definition 2.2 (Taxonomy Expansion). Given 1) an existing taxonomy $\mathcal{T}_0 = (\mathcal{V}_0, \mathcal{E}_0)$, 2) a text corpus $\mathcal{D}$, and 3) a set of candidate terms C, the goal of taxonomy expansion is to insert the term $q \in C$ into the existing taxonomy $\mathcal{T}_0$ and expand it into a more complete taxonomy $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \mathcal{V}_0 \cup C, \mathcal{E} = \mathcal{E}_0 \cup \mathcal{R}$ with $\mathcal{R}$ being the newly discovered relations between terms in C and $\mathcal{V}_0$.

3 THE STEAM METHOD

In this section, we describe our proposed method STEAM. We first give an overview of our method, and then detail *mini-path*-based prediction and multi-view co-training. Finally, we discuss model learning and inference.

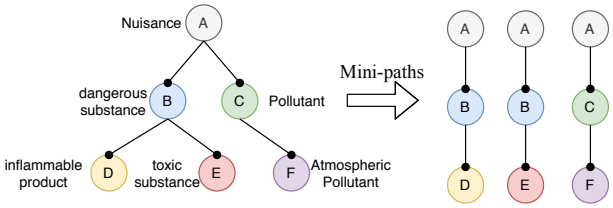
3.1 Self-Supervised Learning by Mini-Path Attachment

The central task of taxonomy expansion is to attach a query term $q \in C$ into the correct position in the existing taxonomy $\mathcal{T}_0$. STEAM is a self-supervised learning procedure, which learns to attach query terms using *natural* supervision signals from the seed taxonomy itself. It creates a prediction task that pinpoints the anchor positions for the terms already in the seed taxonomy. The training data for this self-supervised learning task can be easily obtained from the seed taxonomy without extra annotated hypernymy pairs.

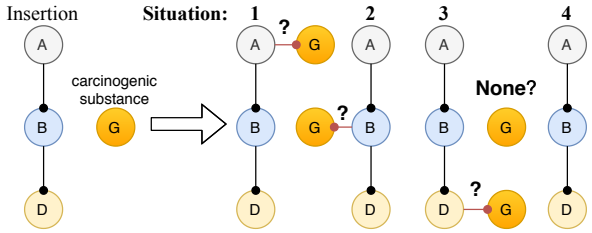
3.1.1 Query-Anchor Matching with Mini-Paths. To instantiate the self-supervised learning paradigm, STEAM learns to match query

terms with anchor structures in the seed taxonomy named *mini-paths*. The mini-paths are snippet paths sampled from the seed taxonomy, which contain terms from different layers to preserve the structural information of the seed taxonomy. Below, we define *mini-path* and formulate the self-supervised learning task based on mini-paths.

Definition 3.1 (Mini-path). A mini-path $P = [p_1, p_2, \ldots, p_L]$ consists of several terms $\{p_1, p_2, \ldots, p_L\} \subset \mathcal{V}_0$, where L is the length of P. Each term pair $\langle p_i, p_{i+1} \rangle$ $(1 \le i \le L-1)$ corresponds to an edge in $\mathcal{E}_0$.



(a) An illustration of mini-paths.



(b) The classification target.

Figure 2: An illustration of the proposed mini-paths and the mini-path-based node attachment task.

The mini-paths are fixed-length paths of the existing taxonomy $\mathcal{T}_0$, as shown in Figure 2(a). They serve as anchors for any new query terms, and the self-supervised learning task is to pinpoint the correct position of a query term in the mini-path. As shown in Figure 2(b), given a length-3 mini-path as *anchor* and a new term as *query*, STEAM predicts the probabilities of the *query* being attached to the three terms, or none of them.

Compared with the simple task of binary hypernymy classification, matching query terms with mini-paths has two major advantages: 1) When attaching a query term, considering the terms $p_i \in P$ provides richer information for query attachment than considering each term separately. 2) Compared with the binary classification, this task is more challenging—the matching module needs to judge not only whether q should be matched to P but also which specific position to attach. Learning from this more challenging task allows STEAM to better leverage the structural information of the existing taxonomy.

3.1.2 Sampling Mini-Paths from the Seed Taxonomy. To train a taxonomy expansion model, we sample mini-paths as well as the query terms from the seed taxonomy. We achieve this by randomly sampling mini-paths from the taxonomy, along with positive and negative query terms for each mini-path.

The detailed procedure for training data creation is described as follows. Given one mini-path $P \in \mathcal{P}$ where $\mathcal{P}$ is the collection of all mini-paths in the existing taxonomy, we first generate positive training set $\mathcal{X}^{\text{pos}}$ by sampling all the child terms $a_{i,l} \in A$ of $P \in \mathcal{P}$, where $a_{i,l}$ is the i-th child of the l-th anchor term $p_l \in P$ and A contains all child terms attached to the mini-path, and a positive pair is represented as $X^{\text{pos}}_{i,l} = \langle a_{i,l}, p_j, l \rangle$. Once $\mathcal{X}^{\text{pos}}$ is obtained, we augment the training set by adopting the negative sampling strategy to generate negative set $\mathcal{X}^{\text{neg}}$ by randomly selecting $|\mathcal{X}^{\text{neg}}| = r \times |\mathcal{X}^{\text{pos}}|$ terms $\{n_i\}_{i=1}^{|\mathcal{X}^{\text{neg}}|}$ with sampling ratio r, each constituting a negative pair with one term that is not its parent in an anchor. Since these negative terms do not directly associate with the mini-path P, we assign a relative position $L+1$ for them to indicate no connection exists. Combining $\mathcal{X}^{\text{pos}}$ and $\mathcal{X}^{\text{neg}}$ together we obtain the final training set $\mathcal{X}$.

After obtaining query-anchor pairs, we need to learn a model using such data. Given the set of training pairs $\mathcal{X}$, we denote each pair as $X = \langle q, P, l \rangle \in \mathcal{X}$ where q is the query term, P is the mini-path, and y is the relative position and aim to learn a model $f(q, P|\Theta)$ to identify the correct position (represented by the true label y). The training objective is to optimize the negative log likelihood $\ell = -\sum_X \sum_{i=1}^{L+1} y_i \log \hat{y}_i$ where $\hat{y}$ is the predicted position.

3.2 Multi-View Co-Training with Mini-Paths

Now the question is how to obtain feature representations for each query-anchor pair (q, P).

3.2.1 Multi-View Feature Extraction. STEAM learns representations of query-anchor pairs from three views: (1) *the distributed representation view*, which captures their correlation from pre-trained word embeddings; (2) *the contextual relation view*, which captures their correlation from the sentences where the query term and anchor terms co-occur; and (3) *the lexico-syntactic view*, which captures their correlation from the linguistic similarities between the query and the anchor.

Each of the three views has its own advantages and disadvantages: (1) *Distributed features* have a high coverage over the term vocabulary, but they do not explicitly model pair-wise relations between a query term and an anchor term; (2) *Contextual features* can capture the relation between two terms from their co-occurred sentences, but have limited coverage over term pairs. For example, only less than 15% of hypernym pairs have co-occurred in the scientific corpus of the SemEval dataset; (3) *Lexico-Syntactic features* encode linguistic information between terms and can work well for matched term pairs, but these features are too rigid to cover all the linguistic patterns, and may also have limited coverage.

Given a query term q and an anchor mini-path $P = [p_1, p_2, \cdots, p_L]$, we describe the details of how we obtain feature representations for the query-anchor pair (q, P) from the three views.

(1) Distributed Features. The first view extracts distributed features for both the query q and the anchor mini-path P. For the query term q and the anchor terms in the mini-path P, we use pre-trained BERT embeddings [9] to initialize their distributed representations.

While it is feasible to directly use such initial embeddings for similarity computation, they are learned in an unsupervised way and not discriminative enough for taxonomy expansion. We thus adopt position-enhanced graph attention network (PGAT) [32] to propagate the initial BERT embeddings to compute embedding-based similarities. Let $\mathbf{w}(q, p_l)$ denote the BERT-based and PGAT-propagated embedding similarity between the query term q and an anchor term $p_l \in P$, then we concatenate these embedding-based similarities for the query-anchor pair (q, P):

$$\mathbf{h}_d(q, P) = [\mathbf{w}(q, p_1) \oplus \cdots \oplus \mathbf{w}(q, p_L)]. \tag{1}$$

(2) Contextual Features. When two terms co-occur in the same sentence, the contexts of their co-occurrence can often indicate the relation between them. Our second view thus harvests the sentences from the given corpus D to extract features for the query term q and the mini-path P. Given the query term q and any anchor term $p_l \in P$, we fetch all the sentences where q and p_l have co-occurred from corpus D. Similar to [35], we process these sentences to extract the dependency paths between q and p_l in these sentences, denoted as $\mathcal{D}_{q,p_l}$. For each dependency path $d_{q,p_l} \in \mathcal{D}_{q,p_l}$, we use an LSTM encoder to learn its representation, denoted as $\mathbf{d}(q, p_l)$. The final contextual features between q and P is thus given by

$$\mathbf{h}_c(q, P) = [\mathbf{d}(q, p_1) \oplus \cdots \oplus \mathbf{d}(q, p_L)]. \tag{2}$$

The details for encoding the dependency-path-based feature is as follows. For each dependency path $d_{q,p_l} \in \mathcal{D}_{q,p_l}$, it is a sequence of context words that lead q to p_l in the dependency tree:

$$d_{q,p_l} = \{v_{e_1}, v_{e_2}, \cdots, v_{e_k}\}, \tag{3}$$

where k is the length of the dependency path. Each edge v_e in the dependency path contains 1) the connecting term v_l, 2) the part-of-speech tag of the connecting term v_{pos}, 3) the dependency label v_{dep}, and 4) the edge direction between two subsequent terms v_{dir}. Formally, each edge v_e is represented as: $v_e = [v_l, v_{pos}, v_{dep}, v_{dir}]$. Now in order to encode each extracted dependency path d_{q,p_l}, we feed the multi-variate sequence d_{q,p_l} into an LSTM encoder. The representation of the LSTM's last hidden layer, denoted as $\text{LSTM}(d_{q,p_l})$, is then used as the representation the path d_{q,p_l}. As the set $\mathcal{D}(q, p_l)$ contains multiple dependency paths between q and p_l, we aggregate them with the attention mechanism to compute the weighted average of these path representations:

$$\begin{aligned} \hat{\alpha}_d &= \mathbf{u}^T \tanh\left(\mathbf{W} \cdot \text{LSTM}(d_{q,p_l})\right), \\ \alpha_d &= \frac{\exp\left(\hat{\alpha}_d\right)}{\sum_{d' \in \mathcal{D}_{q,p_l}} \exp\left(\hat{\alpha}_{d'}\right)}, \\ \mathbf{d}(q, p_l) &= \sum_{d \in \mathcal{D}(q,p_l)} \alpha_d \cdot \text{LSTM}(d_{q,p_l}), \end{aligned} \tag{4}$$

where α_d denotes attention weight for the dependency path d_{q,p_l}; $\mathbf{W}, \mathbf{u}$ are trainable weights for the attention network.

(3) Lexical-Syntactic Features. Our third view extracts lexical-syntactic features between terms. Such features encode the correlations between terms based on their surface string names and syntactic information [23, 27, 42]. Given a term pair (x, y), we extract seven lexical-syntactic features between them as the follows:

- **Ends with**: Identifies whether y ends with x or not.
- **Contains**: Identifies whether y contains x or not.
- **Suffix match**: Identifies whether the k-length suffixes of x and y match or not.
- **LCS**: The length of longest common substring of term x and y.
- **Length Difference**: The normalized length difference between x and y. Let the length of term x and y be $L(x)$ and $L(y)$, then the normalized length difference is calculated as $\frac{|L(x)-L(y)|}{\max(L(x),L(y))}$.
- **Normalized Frequency Difference**: The normalized frequency of (x, y) in corpus D with min-max normalization. Specifically, follow [13], we consider **two types of** normalized frequency based on the noisy hypernym pairs obtained in [27]: (1) *the normalized frequency difference.* Given a term pair (x, y), their normalized frequency is defined as $nf(x, y) = \frac{freq(x,y)}{\max_{z \in \mathcal{V}} freq(x,z)}$ where $freq(x, y)$ defines the occurrence frequency between term (x, y) in the hypernym pairs given by [27] and $\mathcal{V} = \mathcal{V}_0 \cup \mathcal{C}$ which is all terms in the existing taxonomy and test set. Then the first normalize frequence difference is defined as $f(x, y) = nf(x, y) - nf(y, x)$. (2) *the generality difference.* For term x, the normalized generality score $ng(x) = log(1+h)$, where h is defined as the logarithm of the number of its distinct hyponyms. Then the generality difference of term pair $g(x, y)$ is defined as the difference in generality between (x, y) as $g(x, y) = ng(x) - ng(y)$.

Given the query term q and the mini-path $P = [p_1, p_2, \cdots, p_L]$, we compute the lexico-syntactic features for each pair (q, p_l) $(1 \le l \le L)$, denoted as $\mathbf{s}(q, p_l)$. Then we concatenate the features derived from all the term pairs as the lexical-syntactic features for (q, P):

$$\mathbf{h}_s(q, P) = [\mathbf{s}(q, p_1) \oplus \cdots \oplus \mathbf{s}(q, p_L)]. \tag{5}$$

3.2.2 The Multi-View Co-Training Objective. To aggregate the three views for the query-anchor matching, a simple way is to stack three different sets of features and train one unified classifier. However, such feature-level integration can lead to suboptimal results due to two reasons: (1) one view can provide dominant signals over the other two, making it hard to fully unleash the discriminative power of each view; (2) the three views can have different dimensionality and distributions, making learning a unified classifier from concatenated features difficult.

We propose a multi-view co-training procedure (Figure 3) to fuse the three views. It uses the three views to learn three different classifiers and then derives an aggregated classifier from the three classifiers and also encourages their predictions to be consistent. The entire model can be trained in an end-to-end manner. Below, we first describe the base classifiers designed for the three different views and then present the co-training objective.

Base Classifiers from Multiple Views. Based on three sets of feature $\mathbf{h}_d, \mathbf{h}_c, \mathbf{h}_s$ derived from different views, we design three neural classifiers for the query-anchor matching task, *i.e.*, the multi-class classification problem formulated in Section 3.1. For each of the three views, we use a multi-layer perceptron (MLP) with one hidden layer for this prediction task, denoted as f_d, f_s, and f_r. Then the predictions from the three views are given by:

$$\mathbf{y}^d = f_d(\mathbf{h}_d) = \mathbf{W}_2^d(\sigma(\mathbf{W}_1^d\mathbf{h}_d + \mathbf{b}_1^d) + \mathbf{b}_2^d), \tag{6}$$

$$\mathbf{y}^c = f_c(\mathbf{h}_c) = \mathbf{W}_2^c(\sigma(\mathbf{W}_1^c\mathbf{h}_c + \mathbf{b}_1^c) + \mathbf{b}_2^c), \tag{7}$$

$$\mathbf{y}^s = f_c(\mathbf{h}_s) = \mathbf{W}_2^s(\sigma(\mathbf{W}_1^s\mathbf{h}_s + \mathbf{b}_1^s) + \mathbf{b}_2^s), \tag{8}$$

where $\{\mathbf{W}_1^k, \mathbf{W}_2^k, \mathbf{b}_1^k, \mathbf{b}_2^k\}$ $k \in \{d, s, c\}$ are trainable parameters for the three MLP classifiers, and $\sigma(\cdot)$ is the activation function for which we use ReLU in our experiments.

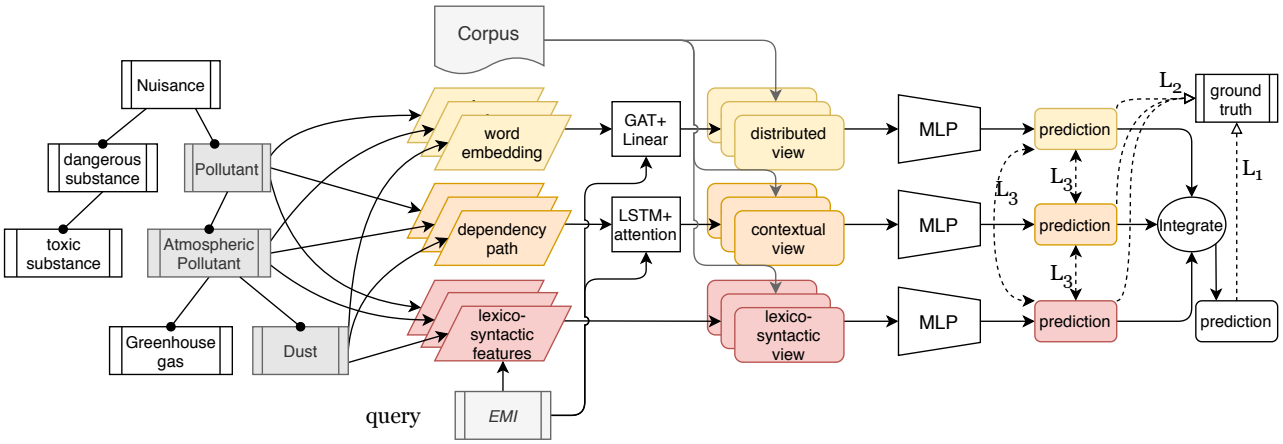


Figure 3: Illustration of the proposed co-training model architecture. The grey terms in the existing taxonomy on the left is an anchor path to attach the new term to. L_1, L_2 and L_3 corresponds to the log-likelihood loss and Euclidean loss calculated in Equation (10), (11) and (12) respectively.

The Co-Training Objective. Figure 3 shows the co-training model that integrates the three base classifiers. From the three base classifiers f_d, f_s, and f_r, we design an aggregated classifier for the final output. This aggregated classifier, which we denote as f_{agg}, integrates the three base classifiers by averaging over their predictions:

$$\mathbf{y}^{\text{agg}} = f_{\text{agg}}\left(\mathbf{y}^d, \mathbf{y}^c, \mathbf{y}^s\right) = \text{softmax}\left(\frac{1}{3}\left(\mathbf{y}^d + \mathbf{y}^s + \mathbf{y}^r\right)\right). \quad (9)$$

To jointly optimize the base classifiers as well as the aggregated classifier, we develop a co-training procedure that not only learns the classifiers to fit the self-supervised signals but also promotes consistency among these classifiers. The co-training objective involves three types of supervision, as detailed below.

The first loss ℓ_1 is defines for the aggregated classifier f^{agg}, which produces the final output. Let $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ be the training dataset, where $\mathbf{x}_i$ is a query-anchor pair and $\mathbf{y}_i$ is the label indicating the correct position of the query term in the anchor mini-path. Then ℓ_1 is defined as the negative log-likelihood loss:

$$\ell_1 = -\sum_{i=1}^{N}\sum_{j=1}^{C} \mathbf{y}_{ij} \log \mathbf{y}_{ij}^{\text{agg}}, \quad (10)$$

where $C = L + 1$ is the number of labels for query-anchor matching.

The second loss ℓ_2 is defined for three base classifiers corresponding to the three views:

$$\ell_2 = -\sum_{u\in\{d,c,s\}}\sum_{i=1}^{N}\sum_{j=1}^{C} \mathbf{y}_{ij} \log \mathbf{y}_{ij}^{u}. \quad (11)$$

The third loss ℓ_3 is a *consistency loss* that encourages the prediction results from different views to agree with each other. We use L2-distance to measure the difference between the classifiers and define ℓ_3 as:

$$\ell_3 = \sum_{u,v\in\{d,s,r\}}\sum_{i=1}^{N} \left\|\mathbf{y}_i^u - \mathbf{y}_i^v\right\|^2. \quad (12)$$

The overall objective of our model is then:

$$\ell = \ell_1 + \lambda\ell_2 + \mu\ell_3, \quad (13)$$

where $\lambda > 0, \mu > 0$ are two pre-defined balancing hyper-parameters.

3.3 Model Learning and Inference

During training, we learn the model parameter Θ by minimizing the total loss ℓ using stochastic gradient optimizers such as Adam [17]. During inference, given a new query term $q \in C$, we traverse all the mini-paths $P \in \mathcal{P}$ and calculate the scores for all anchor terms $p \in P$ based on the aggregated final prediction score $y_{q,p}^P$ in Eq. (9). Specifically, for any anchor term $\hat{p}$, we calculate its score of being the parent of query q as

$$y_{\hat{p}} = \frac{1}{|\hat{\mathcal{P}}|}\sum_{P\in\hat{\mathcal{P}}} y_{q,\hat{p}}^P, \quad (14)$$

where $\hat{\mathcal{P}}$ is the set of mini-paths which contain term $\hat{p}$. Then, we rank all anchor terms and select the term p^* with the highest score as the predicted parent of the query q:

$$p^* = \arg\max_{p\in\mathcal{V}_0} y_p. \quad (15)$$

4 EXPERIMENTS

4.1 Experiment Setup

4.1.1 Datasets. We evaluate the performance of our taxonomy construction method using three public benchmarks. These datasets come from the shared task of taxonomy construction in SemEval 2016 [6]. We use all the three English datasets in SemEval 2016, which correspond to three human-curated concept taxonomies from different domains: environment (EN), science (SCI), and food (Food). For each taxonomy, we start from the root term and randomly grow in a top-down manner until 80% terms are covered. We use the randomly-growed taxonomies as seed taxonomies for self-supervised learning, and the rest 20% terms as our test data. STEAM and several baselines require text corpora for model learning. The details of our used corpora and the matching statistics are described in the Appendix A.

4.1.2 Baselines. We compare with the following baselines:

- **TAXI** [27] is a taxonomy induction method that reached the first place in the SemEval 2016 task. It first extracts hypernym pairs based on substrings and lexico-syntactic patterns with domain-specific corpora and then organizes these terms into a taxonomy.
- **HypeNet** [35] is a strong hypernym extraction method, which uses an LSTM model to jointly model the distributional and relational information between term pairs.
- **BERT+MLP** is a distributional method for hypernym detection based on pre-trained BERT embeddings. For each term pair, it first obtains term embeddings from a pre-trained BERT model and then feeds them into a Multi-layer Perceptron to predict whether they have the hypernymy relationship[3].
- **TaxoExpan** [32] is the state-of-the-art self-supervised taxonomy expansion method. It adopts graph neural networks to encode the positional information and uses a linear layer to identify whether the candidate term is the parent of the query term. For a fair comparison, we also use BERT embeddings for TaxoExpan instead of the word embeddings as in the original paper.

4.1.3 Variants of STEAM. We also compare with several variants of STEAM to evaluate the effectiveness of its different modules: Concat directly concatenates the three features and feeds it into an MLP for prediction; Concat-D concatenates only the context and lexico-syntactic views; Concat-C concatenates the distributed and the lexico-syntactic features; Concat-L concatenates the distributed and the context features; STEAM-Co directly uses the aggregated classifier for prediction instead of the co-training objective (*i.e.*, $\lambda = \mu = 0$); STEAM-D co-trains without the distributed view; STEAM-C co-trains without the contextual view and STEAM-L co-trains without the lexico-syntactic view.

4.1.4 Implementation Details. All the baseline methods, except for BERT-MLP, are obtained from the code published by the original authors. The others (BERT-MLP, our model, and its variants) are all implemented in PyTorch. When learning our model, we use the ADAM optimizer [17] with a learning rate of 1e-3. On all the three datasets, we train the model for 40 epochs as we observe the model has converged after 40 epochs. To prevent overfitting, we used a dropout rate of 0.4 and a weight decay of 5e-4. For encoding context features, we follow [35] and set the dimensions for the POS-tag vector, dependency label vector and edge direction vector to 4, 5, and 1, respectively; and set the dimension for hidden units in the LSTM encoder to 200. For three base MLP classifiers, we set the dimensions of the hidden layers to 50. For sampling negative mini-paths, we set the size of negative samples $r = 4$. In the co-training module, there are two key hyper-parameters: λ and μ for controlling the strength for training base classifiers and the consistency among classifiers. By default, we set $\lambda = 0.1, \mu = 0.1$. We will study how these parameters affect the performance of our model later.

4.1.5 Evaluation Protocol. At the test time, pinpointing the correct parent for a query term is a ranking problem. Follow existing works [22, 32, 36], we use multiple metrics including (1) Accuracy (Acc); (2) Mean reciprocal rank (MRR); (3) Wu & Palmer accuracy (Wu&P) for evaluating the expansion performance.

[3]For combining term embeddings, we experiment with Concat, Difference, and Sum as different fusing functions and report the best performance.

Given n test samples, let us use $\{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n\}$ to denote their ground truth positions, $\{y_1, y_2, \cdots, y_n\}$ to denote model predictions. The metrics we use are computed as follows:
(1) **Accuracy (Acc)** measures the exact match accuracy for terms in the test set. It only counts the cases when the prediction equals to the ground truth, calculated as

$$\text{Acc} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(y_i = \hat{y}_i).$$

(2) **Mean reciprocal rank (MRR)** is the average of reciprocal ranks of a query concept's true parent among all candidate terms. Specifically, it is calculated as

$$\text{MRR} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{rank(y_i)}.$$

(3) **Wu & Palmer similarity (Wu&P)** calculates the semantic similarity between the predicted parent term y and the ground truth parent term $\hat{y}$ as

$$\omega(\hat{y}, y) = \frac{2 \times \text{depth}(\text{LCA}(\hat{y}, y))}{\text{depth}(\hat{y}) + \text{depth}(y)}$$

where "depth$(\cdot)$" is the depth of a term in the taxonomy and "LCA$(\cdot,\cdot)$" is the least common ancestor of the input terms in the taxonomy. Then, the overall Wu&P score is the mean Wu & Palmer similarity for all terms in the test set written as $\text{Wu\&P} = \frac{1}{n}\sum_{i=1}^{n}\omega(y_i, \hat{y}_i)$.

4.2 Experimental Results

4.2.1 Comparison with Baselines. Table 1 reports the performance of STEAM and the baseline methods on the three benchmarks.

Table 1: Comparision of STEAM against the baseline methods on the three datasets (in %). To reduce randomness, we ran all methods for three times and report the average performance. TAXI outputs an entire taxonomy instead of ranking lists, so we are unable to obtain its MRRs.

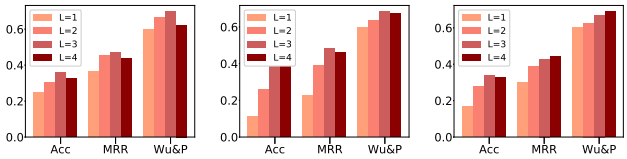
Dataset	Environment			Science			Food		
Metric	Acc	MRR	Wu&P	Acc	MRR	Wu&P	Acc	MRR	Wu&P
BERT+MLP	11.1	21.5	47.9	11.5	15.7	43.6	10.5	14.9	47.0
TAXI	16.7	–	44.7	13.0	–	32.9	18.2	–	39.2
HypeNet	16.7	23.7	55.8	15.4	22.6	50.7	20.5	27.3	63.2
TaxoExpan	11.1	32.3	54.8	27.8	44.8	57.6	27.6	40.5	54.2
STEAM	**36.1**	**46.9**	**69.6**	**36.5**	**48.3**	**68.2**	**34.2**	**43.4**	**67.0**

From the results, we have the following observations:
• STEAM consistently outperforms all the baselines by large margins on the three datasets. In particular, STEAM improves the performance of the state-of-the-art TaxoExpan model by 11.6%, 7.0% and 9.4% for Acc, MRR and Wu&P on average. Such improvements are mainly due to the mini-path-based prediction and the multi-view co-training designs in STEAM.
• Pre-trained BERT embeddings have remarkable expressive power. However, BERT embeddings alone can yield limited performance in the taxonomy expansion task since BERT does not well capture the contextual relations between terms.

• TAXI underperforms other methods on all three datasets. The major drawback of TAXI and other taxonomy construction methods is that they fail to use self-supervision signals in the existing taxonomy. This hinders them from learning the hierarchical and semantic information. Moreover, they simply use lexico-syntactic patterns and neglect other distributional features, which is important for taxonomy expansion.
• HypeNet outperforms BERT and TAXI since it combines the contextual and distributed features. However, it neglects the structural information during training and does not consider lexico-syntactic features, rendering it less effective than STEAM.

4.2.2 Ablations Studies. We perform ablation studies to study the effectiveness of the different components in STEAM: 1) mini-path-based self-supervised learning; 2) the multi-view information; and 3) the co-training procedure.

The Effect of Mini-Paths. To study the effectiveness of mini-path-based self-supervised expansion, we vary the length L of mini-paths. Note that, when $L = 1$, the model is reduced to performing hypernymy prediction. Figure 4 shows the performance of STEAM on the three datasets when L varies. Generally, when L is small, the performance of STEAM stably increases with L. Such results show that mini-paths can effectively capture the structural information in the seed taxonomy—apart from the 'parent' of the query term, the grandparents and siblings contain additional information to improve expansion performance. The mini-paths connect terms from different layers of the taxonomy and carry such information to make the model pinpoint the correct position. However, when L increases from 3 to 4, we observe slight performance drops. This is because the size of the training data shrinks for smaller taxonomies when L becomes larger. Take the environment dataset as an example: It contains 185 training samples when $L = 3$ while 83 when $L = 4$. As a result, the final performance decreases by 3.2% for accuracy.



(a) Environment (b) Science (c) Food

Figure 4: The result for different length of mini-paths L over three datasets.

The Effect of Multi-view Information. We study the contributions of different views by comparing STEAM with its variants (STEAM-D, STEAM-C, STEAM-L). Table 2 shows the results on the three datasets. As shown, it is clear that all three types of features contribute significantly to the overall performance. When eliminating one of the three views, the average performance drops by 6.07%, 8.10% and 4.67% for the three metrics.

The Effect of Co-training. Now we proceed to study the effectiveness of the co-training procedure. While integrating multiple views is important, *how* to integrate multi-view information is equally important. From the results in Table 2, one can see STEAM outperforms Concat by 15.3%, 16.2% and 13.3% for three metrics on average. This verifies the effectiveness of co-training compared with concatenation: the simple concatenation strategy cannot fully harvest the information from each view and could make the learning problem more difficult. Interestingly, the performance for Concat is even worse than Concat-D and Concat-C in accuracy on Food and Environment, which implies that simple concatenation can even hurt the performance with more views.

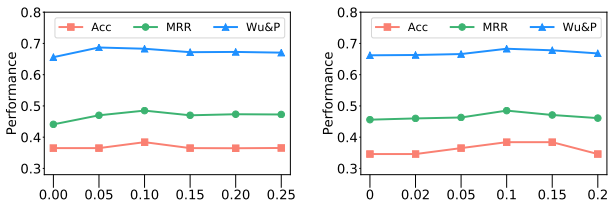
Table 2: Overall results of all variants of our methods on three datasets (in %).

Dataset	Environment			Science			Food		
Metric	Acc	MRR	Wu&P	Acc	MRR	Wu&P	Acc	MRR	Wu&P
Concat	25.0	40.3	64.2	20.4	25.8	51.1	15.5	23.8	49.6
Concat-D	30.6	38.6	63.7	11.1	20.1	48.1	23.1	28.9	55.4
Concat-C	27.7	37.4	57.8	13.5	25.7	53.3	25.3	31.2	58.3
Concat-L	11.1	31.4	57.7	13.5	23.7	39.1	8.30	13.4	40.1
STEAM-Co	25.0	41.0	66.3	32.7	45.3	64.4	31.1	40.7	65.1
STEAM-D	13.8	32.0	54.3	23.1	32.9	60.0	20.1	31.5	60.8
STEAM-C	11.1	26.8	49.2	32.7	44.5	67.2	19.3	29.7	59.3
STEAM-L	11.1	27.5	51.6	23.1	36.5	62.1	12.7	22.6	56.7
STEAM	**36.1**	**46.9**	**69.6**	**36.5**	**48.3**	**68.2**	**34.2**	**43.4**	**67.0**

The co-training objective in STEAM involves two loss terms that encourage better learning of the base classifiers and the consistency among them. From Table 2, the performance gap between STEAM and STEAM-Co shows the effectiveness of these two terms. STEAM-Co only uses the aggregated classifier for prediction and underperforms STEAM by large margins. The reason is that these terms explicitly require *every* base classifier is sufficiently trained and mutually enhances each other; without them, certain views may not be fully leveraged, which limit the effectiveness in leveraging multi-view information for training.

4.2.3 Parameter Studies. In this subsection, we study the effect of different parameters on the performance of STEAM. We have already studied the effect of the path length in the ablation study, now we study the effects of two key parameters in the co-training procedure: 1) the weight of the prediction loss of the three base classifiers λ, and 2) the weight of the consistency loss μ. When evaluating one parameter, we fix other parameters to their default values and report the results. Due to the space limit, we only report the results on parameters on Science dataset as the tends and findings are similar for the three datasets.

Effect of λ. Figure 5(a) shows the effect of λ on the Science dataset. We can observe that as λ increases, the performance improves for all three metrics. This is because larger λ will add more weight to learning base classifiers and enforce each base classifier to achieve good prediction performance. As the base classifiers become stronger, the derived aggregated classifier can also become stronger. However, when $\lambda \geq 0.15$, the performance decreases with λ. We suspect the reason is each single view can be one-sided and noisy to yield biased predictions, when λ is too large, the biased information from each single view can no longer be effectively eliminated during integration, which can hurt the overall performance.



(a) λ (b) μ

Figure 5: The performance of our model when varying different parameters.

Gold Parent: Physics

View	Score	Rank
Distributed	0.812	11
Contextual	0.947	12
Lexico-syntactic	0.640	15
STEAM Output	**0.799**	**1**

(a) term **Electrostatics** (SCI)

Gold Parent: Fruit Juice

View	Score	Rank
Distributed	0.720	25
Contextual	0.921	14
Lexico-syntactic	0.656	15
STEAM Output	**0.765**	**1**

(b) term **Nectar** (Food)

Gold Parent: Mammal

View	Score	Rank
Distributed	0.416	116
Contextual	0.987	1
Lexico-syntactic	0.615	31
STEAM Output	**0.672**	**1**

(c) term **Whale Marine** (EN)

Gold Parent: Medicine

View	Score	Rank
Distributed	0.741	51
Contextual	0.959	2
Lexico-syntactic	0.614	14
STEAM Output	**0.771**	**1**

(d) term **Podiatry** (SCI)

Gold Parent: Red Wine

View	Score	Rank
Distributed	0.468	169
Contextual	0.493	24
Lexico-syntactic	0.329	228
STEAM Output	**0.430**	**43**

(e) term **Chianti** (Food)

Gold Parent: Sea Bed

View	Score	Rank
Distributed	0.387	35
Contextual	0.568	22
Lexico-syntactic	0.483	127
STEAM Output	**0.479**	**37**

(f) term **Inshore Grounds** (EN)

Figure 6: Prediction result for several test terms from different datasets.

Effect of μ. Figure 5(b) shows the effect of μ. Similarly, as μ increases, the performance of STEAM first increases and then decreases when μ is too large. The reason for this phenomenon is that: 1) when μ is too small, the three models cannot regularize each other well, which hinders them from sharing the result with others; 2) when the μ is too large, then the output will be close to optimizing Equation 13. When one model does not perform well, it will negatively affect the other two models, which will deteriorate the performance of the overall model.

4.2.4 Case Studies and Error Analysis. Figure 6 shows multiple cases to illustrate the efficacy of STEAM. It reports the final prediction score of STEAM for the ground-truth parent, as well as the prediction scores from the three base classifiers. Based on the scores, we calculate the rank of the ground truth parent. From Figure 6(a), (b), we can find that there are cases when the predictions from all the three views are inadequate, but the final prediction can integrate the weak signals to rank the ground-truth to the top. Such cases verify the power of multi-view co-training in STEAM, which can utilize the complementary signals from all views and improve the final performance. Besides, Figure 6(c), (d) shows two cases when the predictions of one specific view are poor (*e.g.* Distributed view for term `Whale Marine`), yet STEAM can rectify the mistakes by leveraging the information from the other two views. Figure 6(e) and (f) show two random examples on which our model fails to provide the correct predictions. In such cases, the information from the three views is insufficient to capture the hypernymy relationships between the test term and its parent.

5 RELATED WORK

Taxonomy Construction. There have been many studies on automatic taxonomy construction. One line of works constructs taxonomies using cluster-based methods. They group terms into a hierarchy based on hierarchical clustering [1, 31, 41] or topic models [10, 21]. These methods can work in an unsupervised way. However, they cannot be applied to our taxonomy expansion problem, because they construct *topic-level* taxonomies where each node is a collection of topic-indicative terms instead of single terms. More relevant to our work are the methods developed for constructing *term-level* taxonomies. Focused on taxonomy induction, these methods organize hypernymy pairs into taxonomies. Graph optimization techniques [3, 8, 13, 18] have been proposed to organize the hypernymy graph into a hierarchical structure, and Mao et al. [23] utilize reinforcement learning to organize term pairs by optimizing a holistic tree metric over the training taxonomies. Very recently, Shang et al. [30] design a transfer framework to use the knowledge from existing domains for generating taxonomy for a new domain. However, all these methods attempt to construct taxonomies *from scratch* and cannot preserve the structure of the seed taxonomy.

Hypernymy Detection. Hypernym detection aims at identifying hypernym-hyponym pairs, which is essential to taxonomy construction. Existing methods for hypernymy detection mainly fall into two categories: *pattern-based* methods and *distributed* methods. *Pattern-based* methods extract hypernymy pairs via pre-defined lexico-syntactic patterns [15, 27, 29]. One prominent work in this branch is the Hearst patterns [15], which extract hypernymy pairs based on a set of hand-crafted *is-a* patterns (*e.g.*, "X is a Y"). Pattern-based methods achieve good precision, but they suffer from low recall [39] and are prone to idiomatic expressions and parsing errors [19]. *Distributed* methods detect hypernymy pairs based on the distributed representations (*e.g.* word embeddings [9, 24, 28]) of terms. For a term pair $\langle x, y \rangle$, their embeddings are used for learning a binary classifier to predict whether it has the hypernymy relation [4, 7, 12, 34]. As embeddings are directly learned from the corpora, distributed methods eliminate the needs of designing hand-crafted patterns and have shown strong performance. However, their performance relies on a sufficient amount of labeled term pairs, which can be expensive to obtain.

Taxonomy Expansion. Taxonomy expansion is less studied than taxonomy construction. Most existing works on taxonomy expansion aims to find new *is-a* relations and insert new terms to their hypernyms. For example, Aly et al. [2] refine existing taxonomy by adopting hyperbolic embeddings [26] to better capture hierarchical lexical-semantic relationships, [33, 36] design various semantic patterns to determine the position to attach new concepts for expanding taxonomies, and Fauceglia et al. [11] use a hybrid method to take advantage of linguistic patterns, semantic web and neural networks for taxonomy expansion. However, the above methods only model the 'parent-child' relations and fail to capture the global structure of the existing taxonomy. To better exploit self-supervision signals, Manzoor et al. [22] study expanding taxonomies by jointly learning latent representations for edge semantics and taxonomy concepts.

Recently, Shen et al. [32] propose position-enhanced graph neural networks to encode the neighborhood information for terms when insert them to the seed taxonomy. However, the above two approaches only consider distributional features such as word embeddings but neglect other types of relationships among terms. Compared with these methods, STEAM is novel in two aspects. *First*, it inserts new terms with mini-path-based classification instead of simple hypernym attachment, which models different layers to better preserve the holistic structure. *Second*, it considers multiple sources of features for expansion and integrates them with a multi-view co-training procedure.

6 CONCLUSION

We proposed STEAM, a self-supervised learning framework with mini-path-based prediction and a multi-view co-training objective. The self-supervised learning nature enables our model to leverage the information in the existing taxonomy without extra labeling efforts. Compared with the traditional node-to-node *query-anchor* pairs, using mini-paths captures more structural information thus facilitates the inference of a query's attachment position. The multi-view co-training objective effectively integrates information from multiple input sources, including PGAT-propagated word embeddings, LSTM-embedded dependency paths and lexico-syntactic patterns. Comprehensive experiments on three benchmarks show that STEAM consistently outperforms all baseline models by large margins, which demonstrates its superiority for taxonomy expansion.

ACKNOWLEDGEMENT

This work was in part supported by the National Science Foundation award IIS-1418511, CCF-1533768 and IIS-1838042, the National Institute of Health award 1R01MD011682-01 and R56HL138415.

REFERENCES

[1] Daniele Alfarone and Jesse Davis. 2015. Unsupervised Learning of an IS-A Taxonomy from a Limited Domain-Specific Corpus. In *IJCAL*. 1434–1441.
[2] Rami Aly, Shantanu Acharya, Alexander Ossa, Arne Köhn, Chris Biemann, and Alexander Panchenko. 2019. Every Child Should Have Parents: A Taxonomy Refinement Algorithm Based on Hyperbolic Term Embeddings. In *ACL*. 4811–4817.
[3] Mohit Bansal, David Burkett, Gerard De Melo, and Dan Klein. 2014. Structured learning for taxonomy induction with belief propagation. In *ACL*. 1041–1051.
[4] Marco Baroni, Raffaella Bernardi, Ngoc-Quynh Do, and Chung-chieh Shan. 2012. Entailment above the word level in distributional semantics. In *EACL*. 23–32.
[5] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *SIGMOD*. ACM, 1247–1250.
[6] Georgeta Bordea, Els Lefever, and Paul Buitelaar. 2016. SemEval-2016 Task 13: Taxonomy Extraction Evaluation (TExEval-2). In *SemEval-2016*. ACL, 1081–1091.
[7] Haw-Shiuan Chang, Ziyun Wang, Luke Vilnis, and Andrew McCallum. 2018. Distributional Inclusion Vector Embedding for Unsupervised Hypernymy Detection. In *NAACL*. 485–495.
[8] Anne Cocos, Marianna Apidianaki, and Chris Callison-Burch. 2018. Comparing constraints for taxonomic organization. In *NAACL*. 323–333.
[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.
[10] Doug Downey, Chandra Bhagavatula, and Yi Yang. 2015. Efficient methods for inferring large sparse topic hierarchies. In *the ACL*. 774–784.
[11] Nicolas Rodolfo Fauceglia, Alfio Gliozzo, Sarthak Dash, Md Faisal Mahbub Chowdhury, and Nandana Mihindukulasooriya. 2019. Automatic Taxonomy Induction and Expansion. In *EMNLP-IJCNLP Demo*. 25–30.
[12] Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu. 2014. Learning semantic hierarchies via word embeddings. In *ACL*. 1199–1209.
[13] Amit Gupta, Rémi Lebret, Hamza Harkous, and Karl Aberer. 2017. Taxonomy induction using hypernym subsequences. In *CIKM*. 1329–1338.
[14] Sanda M Harabagiu, Steven J Maiorano, and Marius A Paşca. 2003. Open-domain textual question answering techniques. *Natural Language Engineering* 9, 3 (2003), 231–267.
[15] Marti A Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *COLING*. ACL, 539–545.
[16] Giannis Karamanolakis, Jun Ma, and Xin Luna Dong. 2020. TXtract: Taxonomy-Aware Knowledge Extraction for Thousands of Product Categories. In *ACL*.
[17] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[18] Zornitsa Kozareva and Eduard Hovy. 2010. A semi-supervised method to learn and construct taxonomies using the web. In *EMNLP*. 1110–1118.
[19] Zornitsa Kozareva, Ellen Riloff, and Eduard Hovy. 2008. Semantic Class Learning from the Web with Hyponym Pattern Linkage Graphs. In *ACL*. 1048–1056.
[20] Carolyn E Lipscomb. 2000. Medical subject headings (MeSH). *Bulletin of the Medical Library Association* 88, 3 (2000), 265.
[21] Xueqing Liu, Yangqiu Song, Shixia Liu, and Haixun Wang. 2012. Automatic taxonomy construction from keywords. In *SIGKDD*. 1433–1441.
[22] Emaad Manzoor, Rui Li, Dhananjay Shrouty, and Jure Leskovec. 2020. Expanding Taxonomies with Implicit Edge Semantics. In *The Web Conference 2020*. 2044–2054.
[23] Yuning Mao, Xiang Ren, Jiaming Shen, Xiaotao Gu, and Jiawei Han. 2018. End-to-end reinforcement learning for automatic taxonomy induction. In *ACL*. 2462–2472.
[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *NIPS*. 3111–3119.
[25] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (Nov. 1995), 39–41.
[26] Maximillian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. In *NIPS*. 6338–6347.
[27] Alexander Panchenko, Stefano Faralli, Eugen Ruppert, Steffen Remus, Hubert Naets, Cédrick Fairon, Simone Paolo Ponzetto, and Chris Biemann. 2016. TAXI at SemEval-2016 Task 13: a Taxonomy Induction Method based on Lexico-Syntactic Patterns, Substrings and Focused Crawling. In *SemEval-2016*. ACL, 1320–1327.
[28] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*. ACL, 1532–1543.
[29] Stephen Roller, Douwe Kiela, and Maximilian Nickel. 2018. Hearst Patterns Revisited: Automatic Hypernym Detection from Large Text Corpora. In *ACL*. 358–363.
[30] Chao Shang, Sarthak Dash, Md Faisal Mahbub Chowdhury, Nandana Mihindukulasooriya, and Alfio Gliozzo. 2020. Taxonomy Construction of Unseen Domains via Graph-based Cross-Domain Knowledge Transfer. In *ACL*. ACL.
[31] Jingbo Shang, Xinyang Zhang, Liyuan Liu, Sha Li, and Jiawei Han. 2020. NetTaxo: Automated Topic Taxonomy Construction from Large-Scale Text-Rich Network. In *The Web Conference*.
[32] Jiaming Shen, Zhihong Shen, Chenyan Xiong, Chi Wang, Kuansan Wang, and Jiawei Han. 2020. TaxoExpan: Self-supervised Taxonomy Expansion with Position-Enhanced Graph Neural Network. In *The Web Conference 2020*. 486–497.
[33] 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 *SIGKDD*. 2180–2189.
[34] Yu Shi, Jiaming Shen, Yuchen Li, Naijing Zhang, Xinwei He, Zhengzhi Lou, Qi Zhu, Matthew Walker, Myunghwan Kim, and Jiawei Han. 2019. Discovering Hypernymy in Text-Rich Heterogeneous Information Network by Exploiting Context Granularity. In *CIKM*. ACM, 599–608.
[35] Vered Shwartz, Yoav Goldberg, and Ido Dagan. 2016. Improving Hypernymy Detection with an Integrated Path-based and Distributional Method. In *ACL*. ACL, 2389–2398.
[36] Nikhita Vedula, Patrick K Nicholson, Deepak Ajwani, Sourav Dutta, Alessandra Sala, and Srinivasan Parthasarathy. 2018. Enriching taxonomies with functional domain knowledge. In *SIGIR*. 745–754.
[37] Denny Vrandečiundefined. 2012. Wikidata: A New Platform for Collaborative Data Collection. In *WWW Companion*. ACM, 1063–1064.
[38] Chi Wang, Marina Danilevsky, Nihit Desai, Yinan Zhang, Phuong Nguyen, Thrivikrama Taula, and Jiawei Han. 2013. A phrase mining framework for recursive construction of a topical hierarchy. In *SIGKDD*. 437–445.
[39] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q Zhu. 2012. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD*. 481–492.
[40] Xiaoxin Yin and Sarthak Shah. 2010. Building taxonomy of web search intents for name entity queries. In *WWW*. 1001–1010.
[41] Chao Zhang, Fangbo Tao, Xiusi Chen, Jiaming Shen, Meng Jiang, Brian Sadler, Michelle Vanni, and Jiawei Han. 2018. Taxogen: Unsupervised topic taxonomy construction by adaptive term embedding and clustering. In *SIGKDD*. 2701–2709.
[42] Hao Zhang, Zhiting Hu, Yuntian Deng, Mrinmaya Sachan, Zhicheng Yan, and Eric Xing. 2016. Learning Concept Taxonomies from Multi-modal Data. In *ACL*. 1791–1801.
[43] Yuchen Zhang, Amr Ahmed, Vanja Josifovski, and Alexander Smola. 2014. Taxonomy discovery for personalized recommendation. In *WSDM*. 243–252.

A DATASET DETAILS

A.1 Statistics of the Benchmarks

Our used benchmarks come from the shared task of taxonomy construction in SemEval 2016 [6]. Table 3 shows the statistics of these three benchmarks.

Table 3: The statistics of the three datasets for evaluation.

Dataset	Environment	Science	Food
# of Terms	261	429	1486
# of Edges	261	452	1576
# of Layers	6	8	8

We also list the number of mini-paths used in self-supervised training for three dataset as:

• **Environment**: There are 202 mini-paths for $L = 1$, 202 for $L = 2$, 185 for for $L = 3$ and 83 for $L = 4$.

• **Science**: There are 362 mini-paths for $L = 1$, 382 for $L = 2$, 390 for for $L = 3$ and 357 for $L = 4$.

• **Food**: There are 1229 mini-paths for $L = 1$, 1310 for $L = 2$, 1205 for $L = 3$ and 1142 for $L = 4$.

A.2 External Sources of Text Corpus

Our STEAM method and several baselines also require external text corpora to model the semantic relations between concept terms. For all the three benchmarks, we collect the following public corpora: 1) the Wikipedia dump[4], 2) the UMBC web-based corpus[5]; 3) the One Billion Word Language Modeling Benchmark[6].

We directly match the terms with the corpus with tools available online (*i.e.* WikiExtractor[7]) and only preserve the sentences that term pairs co-occur. In this way, for each dataset, we obtain a tailored corpus which preserves the co-occurrence between terms. The information for these corpora are summarized as:

• **Environment**: The corpus size is 824MB with 1.51M sentences.

• **Science**: The corpus size is 1.36GB with 2.07M sentences.

• **Food**: The corpus size is 2.00GB with 3.42M sentences.

B FEATURE EXTRACTION WITH BERT

For extracting distributed representations, we use a pre-trained BERT base model[8] to obtain word embeddings for our methods and the baselines.[9] The dimensionality of these BERT embeddings are 768. Based on the BERT embeddings, we follow the settings in [32] and use a graph attention network propagate embeddings over the seed taxonomy structure.

Specifically, the version used in our model is pre-trained uncased BERT-base with 12 transformer encoder blocks, 12 attention heads, 768-dimensional hidden layers and 110M parameters in total[10]. To get the embeddings, sentences are fed into the BERT model, and the 768-dimensional vectors of the last layer at the positions corresponding to the phrases are extracted as regarded as the constructed embedding.

C COMPLEXITY ANALYSIS

At the training stage, our model uses $|\mathcal{P}|$ training instances every epoch and thus scales linearly to the number of mini-paths in the existing taxonomy. From above we have listed the number of mini-paths in our training, and the number of such mini-paths is linear to $O(|\mathcal{V}_0|)$ (*i.e.* the number of terms in the existing taxonomy). At the inference stage, for each query term, we calculate $L|\mathcal{P}|$ matching scores, where L is the length of the mini-path. To accelerate the computation, we use GPU for matrix multiplication and pre-calculate distributional and lexico-syntactic features and store the dependency paths for faster evaluation.

D BASELINE SETTINGS

We implement the baselines based on the GitHub Repositories released by the authors except for BERT+MLP which we obtain the BERT embeddings of tokens and feed them into a 2 layer MLP for training. We list the GitHub Repository information for other baselines as follows.

• **TAXI**: https://github.com/uhh-lt/taxi.

• **HypeNet**: https://github.com/vered1986/HypeNET.

• **TaxoExpan**: https://github.com/mickeystroller/TaxoExpan.

[4]We use the 20190801 version of wikidump during our experiments.

[5]https://ebiquity.umbc.edu/resource/html/id/351

[6]https://www.statmt.org/lm-benchmark/

[7]https://github.com/attardi/wikiextractor

[8]https://github.com/google-research/bert

[9]We have also tried on pre-trained GloVe [28] and Poincare Embeddings [26] with different dimensions but find that their performances were not as good as BERT embeddings.

[10]The implementation is at https://github.com/huggingface/transformers.