Seed-Guided Fine-Grained Entity Typing in Science and Engineering Domains

Yu Zhang[1*], Yunyi Zhang[1*], Yanzhen Shen[1],
Yu Deng[2], Lucian Popa[3], Larisa Shwartz[2], ChengXiang Zhai[1], Jiawei Han[1]

[1]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[2]IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA
[3]IBM Almaden Research Center, San Jose, CA, USA
{yuz9, yzhan238, yanzhen4, czhai, hanj}@illinois.edu, {dengy, lpopa, lshwart}@us.ibm.com

Abstract

Accurately typing entity mentions from text segments is a fundamental task for various natural language processing applications. Many previous approaches rely on massive human-annotated data to perform entity typing. Nevertheless, collecting such data in highly specialized science and engineering domains (e.g., software engineering and security) can be time-consuming and costly, without mentioning the domain gaps between training and inference data if the model needs to be applied to confidential datasets. In this paper, we study the task of *seed-guided fine-grained entity typing* in science and engineering domains, which takes the name and a few seed entities for each entity type as the only supervision and aims to classify new entity mentions into both seen and unseen types (i.e., those without seed entities). To solve this problem, we propose SETYPE which first enriches the weak supervision by finding more entities for each seen type from an unlabeled corpus using the contextualized representations of pre-trained language models. It then matches the enriched entities to unlabeled text to get pseudo-labeled samples and trains a textual entailment model that can make inferences for both seen and unseen types. Extensive experiments on two datasets covering four domains demonstrate the effectiveness of SETYPE in comparison with various baselines. Code and data are available at: https://github.com/yuzhimanhua/SEType.

Introduction

Entity typing, i.e., automatically determining the types of entity mentions given their contexts, is a fundamental step for various text mining and natural language processing (NLP) tasks, such as entity linking (Wu et al. 2020) and text classification (Hu et al. 2019). Entity typing in science and engineering domains poses substantial new challenges, calling for dedicated research. First, *fine-grained entity typing is critical for domain-specific applications*. For example, in software and security domains (e.g., StackOverflow threads, GitHub README files, and vulnerability descriptions), entities need to be typed in fine-grained scale (e.g., devices, operating systems, versions, and functions) in order to drive downstream applications (e.g., technical question answering (Yu et al. 2020, 2021) and knowledge graph construction (Rukmono and Chaudron 2023)).

Second, *massive human annotation is too costly to be a solution.* Although entity typing has been extensively studied in NLP, most existing approaches rely on massive human-annotated training data, which are time-consuming and costly to obtain, especially in specialized technical domains. Moreover, practitioners in these domains often need to apply the model to their confidential datasets (e.g., internal software issue reports) which cannot be accessed by external annotators, incurring domain gaps between training and inference data. To alleviate annotation efforts, recent studies explore the setting of few-shot entity typing (Ding et al. 2022; Huang, Meng, and Han 2022; Dai and Zeng 2023), where a few manually labeled samples are provided to train the model. However, unless the entity types are balanced (which is usually not the case – the number of APPLICATION entities is 24 times more than the number of ALGORITHM entities in the StackOverflowNER dataset (Tabassum et al. 2020)), one needs to sample and annotate a much larger number of entities to cover the minority types.

Third, *domain-agnostic zero-shot learning methods do not provide a good solution either.* Previous zero-shot entity typing models (Zhou et al. 2018; Obeidat et al. 2019; Zhang et al. 2020a) rely on type names only and do not seek help from any annotated examples. This makes the model unaware of domain-specific knowledge, which may lead to suboptimal performance in highly specialized science and engineering domains. For example, given the sentence "*ListView keep BooleanSparseArray of checked positions (you can get it with method getCheckedItemPositions()).*", if we ask GPT-3.5 Turbo (Ouyang et al. 2022) about the type of "*ListView*" by providing it with type names only, GPT-3.5 Turbo will answer "USER INTERFACE ELEMENT" instead of the correct answer "LIBRARY CLASS".

To strike a balance between few-shot and zero-shot settings, in this paper, inspired by the weakly supervised setting in text classification (Meng et al. 2018; Mekala and Shang 2020), we confine the supervision signals to be type names and a few (e.g., 5) seed entities per type. In comparison with the labeled samples under the few-shot setting, the given entities under our setting are not associated with any context information. For example, the supervision only tells "*ListView*" is a LIBRARY CLASS in general, but no spe-

*Equal Contribution.

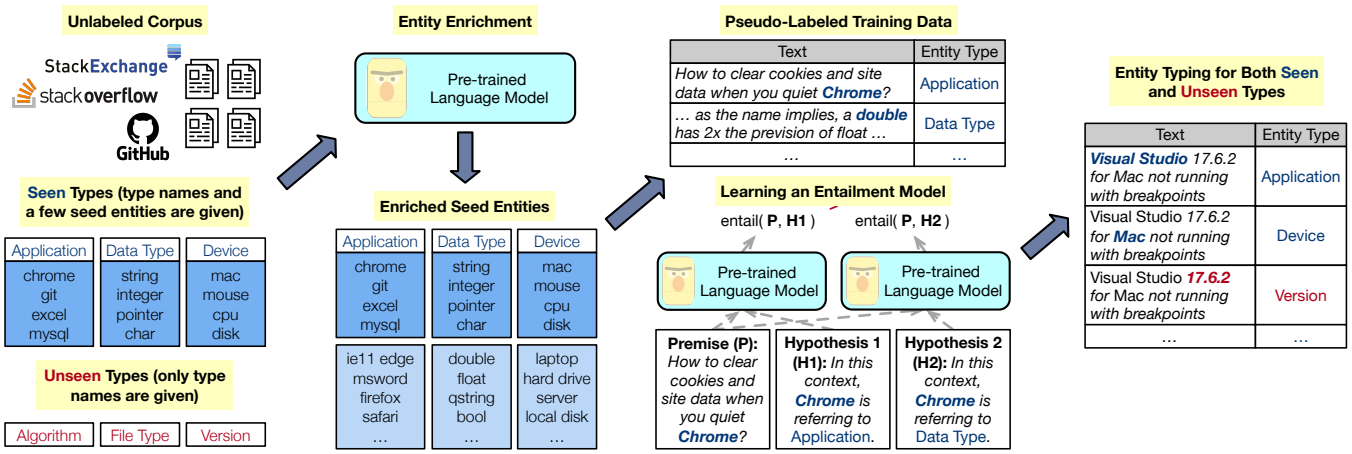


Figure 1: Overview of the SETYPE framework.

cific sentence will be provided. In this case, users just need to name a few entities for each type as supervision, without scanning the text corpus. We call this task setting *seed-guided entity typing*.

Contributions. In this paper, taking software engineering and security as two running domains, we study seed-guided fine-grained entity typing in science and technology. As shown in Figure 1, the task input includes a large unlabeled corpus and some *seen* types. For each seen type, its type name and a small set of seed entities are given. The goal of our task is to train an entity typing model that can classify an entity mention into not only seen types but also *unseen* types. By "unseen", we refer to the types without any seed entities given and never seen during model training. Instead, only their type names are provided during inference. For example, in Figure 1, VERSION is an unseen type, but the model should be able to type "*17.6.2*" as a VERSION entity given its context "*Visual Studio 17.6.2 for Mac not running with breakpoints*".

To perform seed-guided fine-grained entity typing, we propose a framework named SETYPE, which consists of two phases: (i) entity enrichment and (ii) entailment model training. The first phase (i.e., entity enrichment) aims to extract more entities for each seen type from the unlabeled corpus according to their contextualized semantic similarities with the provided seed entities. This is to overcome supervision scarcity. To be specific, if we directly match seed entities to unlabeled text to get pseudo-labeled training samples, the semantic coverage of these obtained samples may still be narrow and cause model overfitting. After finding more entities belonging to each type, the matched training data will be more diverse. Taking such pseudo-labeled data, the model training phase then learns an entailment model by viewing the entity context as a premise and each entity type (filled into a template) as a hypothesis. Finally, the learned model can perform entity typing for both seen and unseen types by predicting to what extent the premise entails the hypothesis corresponding to each candidate type.

To summarize, this study makes the following contributions: (1) *Task*: We propose to study seed-guided fine-grained entity typing. In comparison with the zero-shot setting, it leverages user-provided seed entities as domain knowledge, which is badly needed in highly specialized science and engineering domains. In comparison with the few-shot setting, it alleviates annotation efforts and mitigates domain gaps between training and inference data. (2) *Framework*: We design a two-phase framework, SETYPE, that first conducts entity enrichment to overcome supervision scarcity and then learns an entailment model to perform entity typing for both seen and unseen types. (3) *Experiments*: Extensive experiments on two public datasets (Tabassum et al. 2020; Bridges et al. 2013) covering four domains (i.e., StackOverflow, GitHub, National Vulnerability Database, and Metasploit) demonstrate the effectiveness of SETYPE given 10 to 15 fine-grained types related to code, software, and security. Although we focus on software and security domain examples, our entity typing framework can be applied to other specialized domains including science (Wang et al. 2021) and engineering (O'Gorman et al. 2021).

Problem Definition

Assume there are m entity types $\mathcal{T} = \{t_1, t_2, ..., t_m\}$ (e.g., "DATA STRUCTURE", "DEVICE", "PROGRAMMING LANGUAGE"). For each type t_i, a small set of (e.g., 5) seed entities $\mathcal{E}_i = \{e_{i,1}, ..., e_{i,n}\}$ are given as supervision (e.g., for t_i = "PROGRAMMING LANGUAGE", $\mathcal{E}_i$ = {*c++, java, python, ...*}). The seed-guided entity typing task aims to train a classifier f. Given a text segment d and an entity mention e appears in d, the classifier maps e to its type $f(e|d)$ based on its context. In this paper, we consider two different task settings.

Closed-Set Entity Typing. During inference, the possible type of an entity e always belongs to $\mathcal{T}$. In other words, when making predictions, the model f only needs to consider the entity types it has seen during training.

Open-Set Entity Typing. Following Yuan and Downey (2018), we consider a more challenging setting where some target entity types $\mathcal{T}_u = \{t_{m+1}, t_{m+2}, ..., t_{m+k}\}$ are never seen during training. For each unseen type $t_i \in \mathcal{T}_u$, no seed entity is given as supervision. During inference (i.e., after

the model f is trained), the name of each unseen type (e.g., "VERSION") is given to describe it, and the possible type of an entity e can be either seen or unseen.

Formally, our task is defined as follows.

Definition 1 (Problem Definition) *Given m entity types $\mathcal{T} = \{t_1, t_2, ..., t_m\}$, each of which has its name t_i and a small set of seed entities $\mathcal{E}_i = \{e_{i,1}, ..., e_{i,n}\}$ as input,* ***seed-guided entity typing*** *aims to train a classifier f that maps an entity e mentioned in a text segment d to its type $f(e|d)$. Under the* ***closed-set*** *setting, $f(e|d) \in \mathcal{T}$; under the* ***open-set*** *setting, $f(e|d) \in \mathcal{T} \cup \mathcal{T}_u$, where $\mathcal{T}_u$ is a set of new types with no seed entities given and never seen during training.*

The SETYPE Framework

Since only a few seed entities are provided for each seen type, if we directly match them to an unlabeled corpus to get pseudo-labeled training data, the matched samples may still be scarce and cause model overfitting. Therefore, we propose a two-phase framework, SETYPE (shown in Figure 1), which first enriches each type with more entities and then trains an entailment-based entity typing model with enriched pseudo-labeled samples.

Entity Enrichment

In the first phase, given the sets of seed entities $\mathcal{E}_i = \{e_{i,1}, ..., e_{i,n}\}$ $(1 \le i \le m)$, our goal is to find more entities $\mathcal{E}_i^+ = \{e_{i,n+1}, ..., e_{i,n+l}\}$ that also belong to type t_i from a large unlabeled corpus $\mathcal{D}$. This subtask bears similarities with the entity set expansion task (Rong et al. 2016; Shen et al. 2017; Yu et al. 2019; Zhang et al. 2020c). The difference is that here we need to expand the entity sets of multiple types simultaneously. To be specific, since the expanded entities will be used to match unlabeled corpus to derive pseudo-labeled training data, we would like them to be unambiguous (i.e., always belonging to the same type given different contexts) so that the obtained pseudo labels are likely more accurate. Therefore, we keep mutual exclusivity across different types during expansion (i.e., $(\mathcal{E}_i \cup \mathcal{E}_i^+) \cap (\mathcal{E}_j \cup \mathcal{E}_j^+) = \emptyset,\ \forall i \ne j$).

Given a large corpus $\mathcal{D}$ in a science or engineering domain, we first extract a candidate entity pool $\mathcal{P}$ from $\mathcal{D}$ that will be considered during entity enrichment. Following Zhang et al. (2022b), we adopt AutoPhrase (Shang et al. 2018) to implement this step. Then, we leverage a pre-trained language model **PLM** (e.g., BERTOverflow (Tabassum et al. 2020), which is a BERT model pre-trained on software-related text corpora) to get a representation vector $\boldsymbol{h}_e$ for each seed entity $e \in \bigcup_{i=1}^m \mathcal{E}_i$ or candidate entity $e \in \mathcal{P}$. To achieve this, we find all sentences from $\mathcal{D}$ that contain the entity e. For each such sentence d, following Zhang et al. (2022a), we consider two ways to obtain the contextualized embedding of e: (1) We directly feed d into **PLM**. Note that the entity e may be segmented into multiple tokens by **PLM**. After encoding, each token in d will have an embedding vector, and the embedding of e is the average embedding of tokens in e. (2) We replace e with a `[MASK]` token in d and feed the masked sentence into **PLM**. After encoding, we view the embedding of `[MASK]` as the embedding of e. One can observe that the former approach mentioned above focuses more on the *content* of e, while the latter emphasizes the *context* of e. We concatenate these two embeddings as the sentence-level representation of e in d. Finally, the corpus-level representation of e is the average of all its sentence-level representations in $\mathcal{D}$. Formally,

$$\boldsymbol{h}_e = \frac{1}{\mathrm{sf}(e, \mathcal{D})} \sum_d \Big[\boldsymbol{h}_{e|d}^{\text{content}} \,||\, \boldsymbol{h}_{e|d}^{\text{context}} \Big]. \tag{1}$$

Here, $\mathrm{sf}(e, \mathcal{D})$ is the number of sentences containing e in $\mathcal{D}$ (i.e., "sentence frequency"); $\boldsymbol{h}_{e|d}^{\text{content}}$ and $\boldsymbol{h}_{e|d}^{\text{context}}$ are the sentence-level representations of e in d calculated in the two aforementioned ways.

After obtaining $\boldsymbol{h}_e$, we conduct an iterative entity enrichment process for each type. In each iteration, whether a candidate entity e should be added to $\mathcal{E}_i^+$ is according to the average cosine similarity between e and all entities already belonging to type t_i. Formally,

$$\text{score}(e, t_i) = \frac{1}{|\mathcal{E}_i \cup \mathcal{E}_i^+|} \sum_{e' \in \mathcal{E}_i \cup \mathcal{E}_i^+} \cos(\boldsymbol{h}_e, \boldsymbol{h}_{e'}). \tag{2}$$

At the very beginning of the iterative process, we have $\mathcal{E}_i^+ = \emptyset$ $(1 \le i \le m)$. After each iteration, for each type t_i, we sort all candidate entities according to $\text{score}(e, t_i)$, and the top-ranked entities will be added to $\mathcal{E}_i^+$. To ensure mutual exclusivity, each entity can only be added to its most similar type. For example, if $\text{score}(e, \text{LIBRARY}) = 0.8$ and $\text{score}(e, \text{APPLICATION}) = 0.7$, then e will not be added to the APPLICATION type, even if it is top-ranked according to $\text{score}(e, \text{APPLICATION})$. Equivalently, we define the following score.

$$\text{score}'(e, t_i) = \begin{cases} \text{score}(e, t_i), & \text{if } t_i = \arg\max_t \text{score}(e, t) \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

For each type t_i, the ranking criterion is

$$\max_e \text{score}'(e, t_i), \text{ where } e \in \mathcal{P} \backslash \Big(\bigcup_{j=1}^m (\mathcal{E}_j \cup \mathcal{E}_j^+) \Big). \tag{4}$$

Entailment Model Training

After I iterations of entity enrichment, we have a set of entities $\mathcal{E}_i \cup \mathcal{E}_i^+$ belonging to each type t_i. We use these entities to match an unlabeled corpus. If there is a sentence d in the corpus containing any entity $e \in \mathcal{E}_i \cup \mathcal{E}_i^+$, we create a pseudo-labeled training sample denoted by the triplet (e, d, t_i), which means that an entity e is mentioned in its context d and should be labeled as type t_i. Observing that some sentences in software-related text corpora (e.g., StackOverflow QA threads) are succinct and do not contain sufficient context information, we propose to include the $\pm c$ context sentences of d also as the input to the entity typing model. For the sake of brevity, we use d to denote the text segment consisting of both the sentence mentioning e and its context sentences.

Following Li, Yin, and Chen (2022), we propose to train a natural language inference (NLI) model (Yin, Hay, and

Roth 2019) for entity typing. To be specific, given a pseudo-labeled training sample (e, d, t_i), we treat d as a *premise*, fill e and t_i into a template "*In this context, $\underline{e}$ is referring to $\underline{t_i}$.*" to construct a *hypothesis*, and train the model to recognize that the premise entails the hypothesis. Similar to existing NLI models such as RoBERTa-large-MNLI (Liu et al. 2019) and BART-large-MNLI (Lewis et al. 2020), our model adopts a Cross-Encoder architecture, which concatenates the premise and the hypothesis into one input sequence and feeds it into **PLM**.

$$\boldsymbol{h}_{\texttt{[CLS]}} = \textbf{PLM}(\texttt{[CLS]}\ d\ \texttt{[SEP]}\ \eta(e, t_i)\ \texttt{[SEP]}). \quad (5)$$

Here, $\eta(e, t_i)$ denotes the hypothesis; $\boldsymbol{h}_{\texttt{[CLS]}}$ is the output representation vector of `[CLS]`. We then stack a linear layer upon **PLM** to predict to what extent the hypothesis is correct given the premise.

$$\text{entail}(d, \eta(e, t_i)) = \boldsymbol{w}^\top \boldsymbol{h}_{\texttt{[CLS]}}, \quad (6)$$

where $\boldsymbol{w}$ is a trainable vector.

According to the pseudo label, e belongs to type t_i rather than any other type $t_j \in \mathcal{T}\backslash\{t_i\}$. Therefore, the premise d should entail the hypothesis $\eta(e, t_i)$ and should not entail $\eta(e, t_j)$. To encourage this, we utilize a contrastive loss (Smith and Eisner 2005) during training:

$$\mathcal{J} = -\sum_{t_i \in \mathcal{T}} \sum_{(e,d,t_i)} \sum_{t_j \in \mathcal{T}\backslash\{t_i\}} \log\Big(\frac{\exp(\text{entail}(d, \eta(e, t_i)))}{\exp(\text{entail}(d, \eta(e, t_i))) + \exp(\text{entail}(d, \eta(e, t_j)))}\Big). \quad (7)$$

The model parameters, including **PLM** and $\boldsymbol{w}$, are trained on all pseudo-labeled samples derived by $\mathcal{E}_i \cup \mathcal{E}_i^+ (1 \le i \le m)$ to minimize $\mathcal{J}$.

Inference

After **PLM** and $\boldsymbol{w}$ are trained, given a testing sample (e, d), we are able to predict the type of e. In brief, we enumerate all possible hypotheses to see which one is the most likely to be entailed by d. Under the closed-set setting, the hypothesis space is $\mathcal{H} = \{\eta(e, t_j) | t_j \in \mathcal{T}\}$; under the open-set setting, the hypothesis space becomes $\mathcal{H} = \{\eta(e, t_j) | t_j \in \mathcal{T} \cup \mathcal{T}_u\}$. Formally, for each $\eta(e, t_j) \in \mathcal{H}$, we calculate $\text{entail}(d, \eta(e, t_j))$ according to Eqs. (5) and (6). Then, we pick the type with the highest entailment score as the predicted type of e.

$$f(e|d) = \arg \max_{t_j : \eta(e,t_j) \in \mathcal{H}} \text{entail}(d, \eta(e, t_j)). \quad (8)$$

Experiments

Datasets

We use two publicly available datasets from software engineering and security domains – StackOverflowNER (Tabassum et al. 2020) and Cybersecurity (Bridges et al. 2013).

StackOverflowNER (Tabassum et al. 2020)[1] contain text from two sources – **StackOverflow** question-answer threads and **GitHub** issue reports. We select 10 types as seen types defined in our seed-guided setting, including APPLICATION, DATA STRUCTURE, DATA TYPE, DEVICE, LIBRARY, LIBRARY CLASS, OPERATING SYSTEM, PROGRAMMING LANGUAGE, USER INTERFACE ELEMENT, and WEBSITE, and we pick 5 types as unseen types, including ALGORITHM, FILE TYPE, HTML XML TAG, VALUE, and VERSION. In the original dataset, StackOverflow question-answer threads are split into training, validation, and testing sets, while GitHub issue reports form a testing set only. We take the two testing sets as our testing sets, which are named **StackOverflow** and **GitHub**, respectively. We take the training and validation corpora of StackOverflow, remove their annotations, and treat them as unlabeled corpora to create pseudo-labeled training and validation sets, respectively.

Table 1: Dataset statistics. †: The GitHub dataset does not have any entity annotated as VALUE.

Dataset	StackOverflowNER		Cybersecurity	
	StackOverflow	GitHub	NVD	Metasploit
#Seen types	10	10	5	5
#Unseen types	5	4†	5	5
Average #seeds per seen type	5.4	5.4	5.2	5.2
#Testing samples (Closed-Set)	2,084	3,224	20,798	2,087
#Testing samples (Open-Set)	2,610	3,762	27,646	2,612

Cybersecurity (Bridges et al. 2013)[2] contains text related to the Common Vulnerability Enumeration (CVE) from the National Vulnerability Database (**NVD**) and the **Metasploit** Framework. We select 5 types as seen types – APPLICATION, EDITION, OPERATING SYSTEM, RELEVANT TERM, and VENDOR; 5 other types are treated as unseen types – FILE, FUNCTION, METHOD, PARAMETER, and VERSION. For the larger **NVD** corpus, we take 20% as the annotated testing data, and the remaining 80% are treated as unlabeled text to create pseudo-labeled training and validation data. For the smaller **Metasploit** corpus, we take all annotated samples for testing.

For each seen type, 4-7 seed entities are given. Statistics of the datasets are summarized in Table 1.

The large unlabeled corpus $\mathcal{D}$ for entity enrichment is sampled from the Stack Exchange data dump[3]. $\mathcal{D}$ consists of 1.26 million questions and answers.

Compared Methods

We compare SETYPE with the following baselines.

[1] https://github.com/jeniyat/StackOverflowNER

[2] https://github.com/stucco/auto-labeled-corpus

[3] https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z

Table 2: Performance of compared methods on StackOverflow and GitHub from the StackOverflowNER dataset (Tabassum et al. 2020) as well as NVD and Metasploit from the Cybersecurity dataset (Bridges et al. 2013). Bold: the highest score. *: SETYPE is significantly better than this method with p-value < 0.05. **: SETYPE is significantly better than this method with p-value < 0.01.

	StackOverflow				**GitHub**			
	Closed-Set		Open-Set		Closed-Set		Open-Set	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
RoBERTa-large-MNLI (Liu et al. 2019)	34.74**	31.30**	27.85**	24.57**	34.62**	29.87**	31.55**	25.65**
BART-large-MNLI (Lewis et al. 2020)	30.95**	24.31**	27.20**	21.08**	23.39**	22.09**	24.46**	21.75**
BERTOverflow (Tabassum et al. 2020)	27.59**	25.34**	–	–	20.47**	17.73**	–	–
SetExpan+Entailment (Shen et al. 2017)	48.99**	48.06**	40.96**	35.00**	39.14**	47.16**	35.73**	36.72**
CGExpan+Entailment (Zhang et al. 2020c)	61.37**	59.97*	56.02**	49.43**	45.97**	52.13	47.45*	46.85*
GPT-3.5 Turbo (Ouyang et al. 2022)	50.79**	48.07**	51.86**	49.23**	45.49**	46.37**	49.85	48.42
SETYPE	**66.15**	**64.16**	**60.05**	**52.83**	**52.30**	**55.20**	**52.45**	**49.83**
Fully Supervised	82.77	82.85	73.52	65.62	74.63	77.51	71.66	65.24

	NVD				**Metasploit**			
	Closed-Set		Open-Set		Closed-Set		Open-Set	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
RoBERTa-large-MNLI (Liu et al. 2019)	64.00**	29.34**	52.55**	29.53**	56.06**	26.92**	51.34**	24.93**
BART-large-MNLI (Lewis et al. 2020)	65.43**	20.66**	51.26**	18.12**	59.61**	23.55**	54.42**	21.67**
BERTOverflow (Tabassum et al. 2020)	26.07**	15.46**	–	–	31.77**	20.56**	–	–
SetExpan+Entailment (Shen et al. 2017)	66.86**	**57.07**	51.01**	29.27**	71.70	**60.15**	57.40**	31.52
CGExpan+Entailment (Zhang et al. 2020c)	58.63**	53.10	49.64**	32.35	67.86**	59.48	57.09**	34.61
GPT-3.5 Turbo (Ouyang et al. 2022)	54.71**	46.41**	50.45**	**45.25**	60.04**	50.43**	50.19**	**42.78**
SETYPE	**75.93**	55.34	**61.69**	36.62	**74.77**	57.26	**62.50**	34.72
Fully Supervised	97.67	90.50	74.17	39.47	98.28	82.09	78.87	40.58

RoBERTa-large-MNLI (Liu et al. 2019) is a natural language inference model which is obtained by fine-tuning RoBERTa-large on the MNLI dataset (Williams, Nangia, and Bowman 2018). Li, Yin, and Chen (2022) propose to utilize it as a zero-shot entity typing model by viewing the mention text as a premise and the candidate entity type as a hypothesis.

BART-large-MNLI (Lewis et al. 2020) is obtained by fine-tuning BART-large on MNLI. It can be used as a zero-shot entity typing model in the same way as RoBERTa-large-MNLI.

BERTOverflow (Tabassum et al. 2020) is a base-size BERT model pre-trained on StackOverflow data. We adopt it for few-shot entity typing. To be specific, for each entity mention e appearing in text d, we feed d into BERTOverflow and get the contextualized embedding of e (i.e., $\boldsymbol{h}_{e|d}^{\text{content}}$ in Eq. (1)). Then, for each seen type t_i, we obtain its embedding by feeding its seed entities $\mathcal{E}_i$ into BERTOverflow (without any context) and taking the average of their embeddings. Finally, we select the type that is the most similar to e in the BERTOverflow embedding space (measured by the cosine similarity) as the prediction. Note that this method can only be used for closed-set entity typing because unseen types do not have seed entities.

SetExpan+Entailment first uses SetExpan (Shen et al. 2017) for entity enrichment and then uses enriched entities to create pseudo-labeled training data to train an entailment model. The entailment model is initialized with BERTOverflow. It can be viewed as an ablation version of SETYPE by replacing our entity enrichment step with SetExpan.

CGExpan+Entailment is similar to SetExpan+Entailment, but it replaces our entity enrichment step with CGExpan (Zhang et al. 2020c).

GPT-3.5 Turbo (Ouyang et al. 2022) is a large language model pre-trained on massive corpora with instructions and human feedback. We use it for zero-shot entity typing by inputting the entity e and the context d and asking the model to select an entity type from the candidate type space.

Implementation and Hyperparameters

SETYPE uses BERTOverflow as the **PLM**. During entity enrichment, the number of enriched entities $|\mathcal{E}_i^+|$ is 50 and 100 on StackOverflowNER and Cybersecurity, respectively. In practice, this hyperparameter can be set based on users' knowledge about the rough number of concepts belonging to the types. During model training, the window size of context sentences $c = 1$; the maximum premise length is 462 tokens; the maximum hypothesis length is 50 tokens; the training batch size is 4; we use the AdamW optimizer (Loshchilov and Hutter 2019), warm up the learning rate for the first 100 steps and then linearly decay it, where the learning rate is 5e-5; the weight decay is 0.01, and $\epsilon =$ 1e-8. The model is trained on one NVIDIA RTX A6000 GPU.

Evaluation Metrics

We use Micro-F1 and Macro-F1 scores as evaluation metrics for both closed-set and open-set settings.

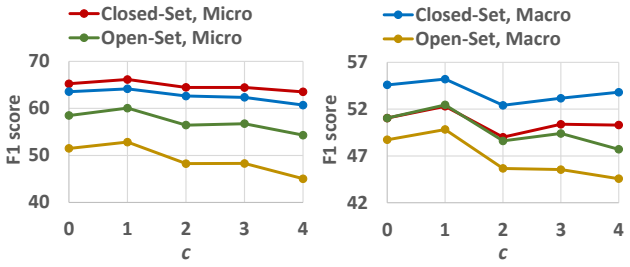
Performance Comparison

Table 2 shows the performance of compared methods on StackOverflowNER and Cybersecurity, respectively. We run SETYPE five times with the average performance reported. To show statistical significance, we conduct a two-tailed Z-test to compare SETYPE with each baseline, and the significance level is also shown in Table 2. We also present the performance of a fully supervised entity typing model, where the ground-truth training and validation sets from StackOverflow and NVD are used to train an entailment model. Note that the term "fully supervised" here corresponds to the closed-set setting rather than the open-set one. In other words, the training and validation sets contain annotations for seen types only, while for unseen types, the model still only has their names during inference.

From Table 2, we can observe that: (1) SETYPE outperforms all baselines significantly in most cases. On StackOverflow and GitHub, SETYPE is consistently the best. On NVD and Metasploit, SETYPE achieves the highest Micro-F1 scores and the second highest Macro-F1 scores. (2) If we do not perform any entity enrichment and directly utilize BERTOverflow for few-shot entity typing, the performance is even lower than the zero-shot RoBERTa-large-MNLI and BART-large-MNLI models in most cases. This is possibly because RoBERTa-large-MNLI and BART-large-MNLI are fine-tuned on general-domain NLI data and enjoy a larger PLM backbone. By contrast, SetExpan+Entailment, CGExpan+Entailment, and SETYPE add an entity enrichment step before using BERTOverflow, and they can outperform RoBERTa-large-MNLI and BART-large-MNLI in many cases. This observation underscores the importance of our proposed entity enrichment phase. (3) SETYPE beats SetExpan+Entailment and CGExpan+Entailment in most cases. Since the entailment model training modules of all these three models are identical, this finding implies that our proposed entity enrichment method is better than SetExpan and CGExpan. The possible reason is that our method is specifically designed for expanding multiple entity sets simultaneously and explicitly models mutual exclusivity across all types.

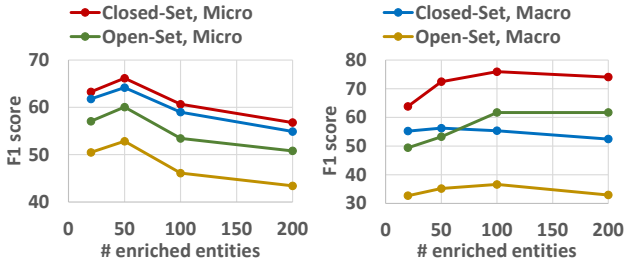
Hyperparameter Analyses

Effect of the Window Size of Context Sentences. As mentioned in our model design, we include the $\pm c$ context sentences of the mention sentence as model input to complement the information in the mention sentence. In SETYPE, we set $c = 1$. Figure 2 depicts the performance of SETYPE with different values of c. We can find that: (1) The F1 scores of SETYPE with $c = 1$ are consistently better than those with $c = 0$. This trend validates our motivation for including context sentences. (2) If we further increase the value of c, the performance starts to fluctuate or even drop. This observation is intuitive because when we stretch too far away, the sentence may be irrelevant to the mentioned entity and bring more noises than hints.



(a) StackOverflow (b) GitHub

Figure 2: Performance of SETYPE with different window sizes of context sentences (the sentence containing the entity mention and its $\pm c$ sentences are fed into **PLM**) on StackOverflow and GitHub. Considering ± 1 sentences is always better than focusing on the mention sentence alone.



(a) StackOverflow (b) NVD

Figure 3: Performance of SETYPE with different numbers of enriched entities per type (i.e., $|\mathcal{E}_i^+|$) on StackOverflow and NVD.

Effect of the Number of Enriched Entities. On StackOverflowNER and Cybersecurity, we extract 50 and 100 entities, respectively, for each seen type to form the enriched entity set $\mathcal{E}_i^+$. We now examine the effect of $|\mathcal{E}_i^+|$ on model performance, which is plotted in Figure 3. We can see that: (1) When we increase $|\mathcal{E}_i^+|$ from 20 to 50, SETYPE consistently performs better. This again implies the positive contribution of finding more entities to tackle supervision scarcity. (2) If we further increase $|\mathcal{E}_i^+|$ from 100 to 200, the performance often drops. This is because lower-ranked entities may not belong to the corresponding type, inducing errors and noises in pseudo-labeled training data. Moreover, extracting more entities will produce more training samples and make training time longer. Overall, we believe setting $|\mathcal{E}_i^+|$ to be 50 or 100 strikes a good balance.

Effect of the Hypothesis Template. In SETYPE, we use the template "*In this context, entity is referring to type.*" to create hypotheses. This template is introduced in Li, Yin, and Chen (2022), where two other templates are also proposed. We show the three templates in Table 3 together with an example premise. Following the terminologies in Li, Yin, and Chen (2022), we name the three templates "Contextual", "Taxonomic", and "Substitution", respectively. The Micro-F1 scores of SETYPE with different templates are also shown in Table 3. We observe that the Substitution template is significantly inferior to the Contextual template; the Taxonomic template performs slightly worse than the Con-

Table 3: Micro-F1 scores of SETYPE with different hypothesis templates used in Li, Yin, and Chen (2022). Bold, *, and **: the same meaning as in Table 2.

Premise: Visual Studio 17.6.2 for **Mac** not running with breakpoints	**StackOverflow**		**GitHub**		**NVD**		**Metasploit**	
	Closed	Open	Closed	Open	Closed	Open	Closed	Open
Contextual Template: In this context, **Mac** is referring to DEVICE.	**66.15**	**60.05**	**52.30**	**52.45**	**75.93**	61.69	**74.77**	62.50
Taxonomic Template: Mac is a DEVICE.	65.29	58.85	51.59	50.63	75.42	**63.64**	73.27	**63.44**
Substitution Template: Visual Studio 17.6.2 for DEVICE not running with breakpoints	57.50**	41.67**	48.36	36.80**	63.57**	47.14**	47.24**	37.23**

textual template in general, but the gap is not significant.

Related Work

Zero-shot and Few-shot Entity Typing

Zero-shot and few-shot entity typing aim to classify a given entity mention to a set of types with limited or no training data. For example, ZOE (Zhou et al. 2018) and DZET (Obeidat et al. 2019) propose to utilize pre-trained word embeddings (i.e., ELMo (Peters et al. 2018) and GLoVe (Pennington, Socher, and Manning 2014), respectively) and Wikipedia's entry descriptions to map entity mentions and types into a shared latent space; MZET (Zhang et al. 2020a) captures semantic meanings and hierarchical information to transfer knowledge from seen to unseen types; PLET (Ding et al. 2022) uses prompt-learning with self-supervision to utilize a PLM's prior knowledge; Huang, Meng, and Han (2022) exploit a PLM's generative power to synthesize new instances for each entity type by feeding a prompt into a PLM and predicting the `[MASK]` tokens; Cui et al. (2022) and Yuan et al. (2022) continue to exploit the power of prompt tuning, while introducing a prototypical verbalizer or adding on prompt and curriculum instructions; Ouyang et al. (2023) exploit ontology structures of entity types and present an ontology enrichment framework for zero-shot entity typing. In addition, the power of transfer learning with PLMs has also been explored. For example, Dai and Zeng (2023) propose to train a general ultra-fine entity typing model and fine-tune it on fine entity typing data. However, in previous studies, context-aware annotated samples (i.e., labeled mentions with the sentence/document they appear in) are given under the few-shot setting. In comparison, in our setting, only a few seed entities are given, without their context, which are weaker supervision signals. In addition to the difference in the level of supervision, the previous studies aim to perform entity typing in the general domain, whereas ours works for specialized technical domains.

Text Mining in Technical Domains

Information extraction and text mining in technical domains such as software engineering and security have been explored to benefit domain-specific applications such as technical question answering (Yu et al. 2020, 2021), knowledge graph construction (Rukmono and Chaudron 2023), and code recommendation (Jin et al. 2023). For named entity recognition (NER) in technical domains, Ye et al. (2016) show that conditional random fields can outperform traditional rule-based models; Tabassum et al. (2020) further leverage the power of PLMs and achieve better performance on StackOverflow and GitHub; Lopez et al. (2021) find that the layout information from raw PDFs can help capture document structure for better NER performance. However, unlike SETYPE, these methods are all limited by their fully supervised settings requiring massive annotated data. In cases where annotations are insufficient or unavailable, Bridges et al. (2013) manage to automatically generate labeled data through matching entries in a security database; Yang et al. (2021) show that high-quality NER labels can be produced by PLMs based on security vulnerability reports. In comparison with the settings of these studies, SETYPE uses only a few seed words as supervision, which alleviates the burden of modeling knowledge bases or annotating training samples. For text classification in technical domains, Zhang et al. (2019, 2020b, 2021) devise weakly supervised approaches that use metadata and/or label hierarchy to classify GitHub repositories. While classification and entity expansion mainly require document or corpus-level understanding of text, the more challenging task of seed-guided entity typing, as described by SETYPE, relies on more fine-grained understanding of the context.

Conclusions and Future Work

In this paper, we study seed-guided fine-grained entity typing in science and engineering domains, which aims to perform entity typing given only type names and a small set of seed entities. We present SETYPE, a two-phase entity typing framework that first conducts entity enrichment and then employs the enriched entities to obtain pseudo-labeled data for subsequent entailment model training. SETYPE is able to classify new entity mentions into both seen and unseen types. With extensive experiments on two public datasets encompassing multiple domains, we demonstrate the significant advantage of SETYPE over zero-shot and seed-guided baselines given 10 to 15 fine-grained types related to code, software, and security. Ablation studies and hyperparameter analyses further validate some key design choices in SETYPE. Interesting future studies include: (1) leveraging large language models to synthesize pseudo-labeled training samples with the help of prompts and (2) exploiting domain-specific knowledge bases to create distant supervision to help fine-grained entity typing.

Acknowledgments

We thank anonymous reviewers for their valuable and insightful feedback. This work was supported by the IBM-Illinois Discovery Accelerator Institute, National Science Foundation IIS-19-56151, and US DARPA INCAS Program No. HR001121C0165.

References

Bridges, R. A.; Jones, C. L.; Iannacone, M. D.; Testa, K. M.; and Goodall, J. R. 2013. Automatic labeling for entity extraction in cyber security. *arXiv preprint arXiv:1308.4941*.

Cui, G.; Hu, S.; Ding, N.; Huang, L.; and Liu, Z. 2022. Prototypical Verbalizer for Prompt-based Few-shot Tuning. In *ACL'22*, 7014–7024.

Dai, H.; and Zeng, Z. 2023. From Ultra-Fine to Fine: Fine-tuning Ultra-Fine Entity Typing Models to Fine-grained. In *ACL'23*, 2259–2270.

Ding, N.; Chen, Y.; Han, X.; Xu, G.; Wang, X.; Xie, P.; Zheng, H.; Liu, Z.; Li, J.; and Kim, H.-G. 2022. Prompt-learning for Fine-grained Entity Typing. In *ACL'22*, 6888–6901.

Hu, L.; Yang, T.; Shi, C.; Ji, H.; and Li, X. 2019. Heterogeneous graph attention networks for semi-supervised short text classification. In *EMNLP'19*, 4821–4830.

Huang, J.; Meng, Y.; and Han, J. 2022. Few-shot fine-grained entity typing with automatic label interpretation and instance generation. In *KDD'22*, 605–614.

Jin, Y.; Bai, Y.; Zhu, Y.; Sun, Y.; and Wang, W. 2023. Code Recommendation for Open Source Software Developers. In *WWW'23*, 1324–1333.

Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; and Zettlemoyer, L. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *ACL'20*, 7871–7880.

Li, B.; Yin, W.; and Chen, M. 2022. Ultra-fine entity typing with indirect supervision from natural language inference. *TACL*, 10: 607–622.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lopez, P.; Du, C.; Cohoon, J.; Ram, K.; and Howison, J. 2021. Mining software entities in scientific literature: document-level ner for an extremely imbalance and large-scale task. In *CIKM'21*, 3986–3995.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *ICLR'19*.

Mekala, D.; and Shang, J. 2020. Contextualized Weak Supervision for Text Classification. In *ACL'20*, 323–333.

Meng, Y.; Shen, J.; Zhang, C.; and Han, J. 2018. Weakly-supervised neural text classification. In *CIKM'18*, 983–992.

Obeidat, R.; Fern, X.; Shahbazi, H.; and Tadepalli, P. 2019. Description-based zero-shot fine-grained entity typing. In *NAACL'19*, 807–814.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. 2022. Training language models to follow instructions with human feedback. In *NeurIPS'22*, 27730–27744.

Ouyang, S.; Huang, J.; Pillai, P.; Zhang, Y.; Zhang, Y.; and Han, J. 2023. Ontology Enrichment for Effective Fine-grained Entity Typing. *arXiv preprint arXiv:2310.07795*.

O'Gorman, T.; Jensen, Z.; Mysore, S.; Huang, K.; Mahbub, R.; Olivetti, E.; and McCallum, A. 2021. MS-Mentions: consistently annotating entity mentions in materials science procedural text. In *EMNLP'21*, 1337–1352.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *EMNLP'14*, 1532–1543.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep Contextualized Word Representations. In *NAACL'18*, 2227–2237.

Rong, X.; Chen, Z.; Mei, Q.; and Adar, E. 2016. Egoset: Exploiting word ego-networks and user-generated ontology for multifaceted set expansion. In *WSDM'16*, 645–654.

Rukmono, S. A.; and Chaudron, M. R. 2023. Enabling Analysis and Reasoning on Software Systems through Knowledge Graph Representation. In *MSR'23*, 120–124.

Shang, J.; Liu, J.; Jiang, M.; Ren, X.; Voss, C. R.; and Han, J. 2018. Automated phrase mining from massive text corpora. *IEEE TKDE*, 30(10): 1825–1837.

Shen, J.; Wu, Z.; Lei, D.; Shang, J.; Ren, X.; and Han, J. 2017. Setexpan: Corpus-based set expansion via context feature selection and rank ensemble. In *ECML-PKDD'17*, 288–304.

Smith, N. A.; and Eisner, J. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *ACL'05*, 354–362.

Tabassum, J.; Maddela, M.; Xu, W.; and Ritter, A. 2020. Code and Named Entity Recognition in StackOverflow. In *ACL'20*, 4913–4926.

Wang, X.; Hu, V.; Song, X.; Garg, S.; Xiao, J.; and Han, J. 2021. ChemNER: fine-grained chemistry named entity recognition with ontology-guided distant supervision. In *EMNLP'21*, 5227–5240.

Williams, A.; Nangia, N.; and Bowman, S. R. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *NAACL'18*, 1112–1122.

Wu, L.; Petroni, F.; Josifoski, M.; Riedel, S.; and Zettlemoyer, L. 2020. Scalable Zero-shot Entity Linking with Dense Entity Retrieval. In *EMNLP'20*, 6397–6407.

Yang, G.; Dineen, S.; Lin, Z.; and Liu, X. 2021. Few-Sample Named Entity Recognition for Security Vulnerability Reports by Fine-Tuning Pre-Trained Language Models. *arXiv preprint arXiv:2108.06590*.

Ye, D.; Xing, Z.; Foo, C. Y.; Ang, Z. Q.; Li, J.; and Kapre, N. 2016. Software-specific named entity recognition in software engineering social content. In *SANER'16*, 90–101.

Yin, W.; Hay, J.; and Roth, D. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *EMNLP'19*, 3914–3923.

Yu, P.; Huang, Z.; Rahimi, R.; and Allan, J. 2019. Corpus-based set expansion with lexical features and distributed representations. In *SIGIR'19*, 1153–1156.

Yu, W.; Wu, L.; Deng, Y.; Mahindru, R.; Zeng, Q.; Guven, S.; and Jiang, M. 2020. A technical question answering system with transfer learning. In *EMNLP'20, System Demonstrations*, 92–99.

Yu, W.; Wu, L.; Deng, Y.; Zeng, Q.; Mahindru, R.; Guven, S.; and Jiang, M. 2021. Technical Question Answering across Tasks and Domains. In *NAACL'21, Industry Papers*, 178–186.

Yuan, S.; Yang, D.; Liang, J.; Li, Z.; Liu, J.; Huang, J.; and Xiao, Y. 2022. Generative Entity Typing with Curriculum Learning. In *EMNLP'22*, 3061–3073.

Yuan, Z.; and Downey, D. 2018. Otyper: A neural architecture for open named entity typing. In *AAAI'18*, 6038–6044.

Zhang, T.; Xia, C.; Lu, C.-T.; and Philip, S. Y. 2020a. MZET: Memory Augmented Zero-Shot Fine-grained Named Entity Typing. In *COLING'20*, 77–87.

Zhang, Y.; Chen, X.; Meng, Y.; and Han, J. 2021. Hierarchical metadata-aware document categorization under weak supervision. In *WSDM'21*, 770–778.

Zhang, Y.; Guo, F.; Shen, J.; and Han, J. 2022a. Unsupervised key event detection from massive text corpora. In *KDD'22*, 2535–2544.

Zhang, Y.; Meng, Y.; Huang, J.; Xu, F. F.; Wang, X.; and Han, J. 2020b. Minimally supervised categorization of text with metadata. In *SIGIR'20*, 1231–1240.

Zhang, Y.; Shen, J.; Shang, J.; and Han, J. 2020c. Empower Entity Set Expansion via Language Model Probing. In *ACL'20*, 8151–8160.

Zhang, Y.; Xu, F. F.; Li, S.; Meng, Y.; Wang, X.; Li, Q.; and Han, J. 2019. Higitclass: Keyword-driven hierarchical classification of github repositories. In *ICDM'19*, 876–885.

Zhang, Y.; Zhang, Y.; Jiang, Y.; Michalski, M.; Deng, Y.; Popa, L.; Zhai, C.; and Han, J. 2022b. Entity Set Co-Expansion in StackOverflow. In *IEEE BigData'22*, 4792–4795.

Zhou, B.; Khashabi, D.; Tsai, C.-T.; and Roth, D. 2018. Zero-Shot Open Entity Typing as Type-Compatible Grounding. In *EMNLP'18*, 2065–2076.