Zero-shot Learning for Named Entity Recognition in Software Specification Documents

Souvick Das
Ca' Foscari University
Venice, Italy
souvick.das@unive.it

Novarun Deb
IIIT Vadodara
Gandhinagar, India
novarun_deb@iiitvadodara.ac.in

Agostino Cortesi
Ca' Foscari University
Venice, Italy
cortesi@unive.it

Nabendu Chaki
University of Calcutta
Kolkata, India
nabendu@ieee.org

2023 IEEE 31st International Requirements Engineering Conference (RE) | 979-8-3503-2689-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/RE57278.2023.00019

***Abstract*—Named entity recognition (NER) is a natural language processing task that has been used in Requirements Engineering for the identification of entities such as actors, actions, operators, resources, events, GUI elements, hardware, APIs, and others. NER might be particularly useful for extracting key information from Software Requirements Specification documents, which provide a blueprint for software development. However, a common challenge in this domain is the lack of annotated data. In this article, we propose and analyze two zero-shot approaches for NER in the requirements engineering domain. These are found to be particularly effective in situations where labeled data is scarce or non-existent. The first approach is a template-based zero-shot learning mechanism that uses the prompt engineering approach and achieves 93% accuracy according to our experimental results. The second solution takes an orthogonal approach by transforming the entity recognition problem into a question-answering task which results in 98% accuracy. Both zero-shot NER approaches introduced in this work perform better than the existing state-of-the-art solutions in the requirements engineering domain.**

Keywords — ***Zero-shot Learning, Named entity recognition, Prompt Engineering, Natural Language Inference***

I. Introduction

Software Requirements Specifications (SRS) are essential blueprints for software development, outlining scope, budget, use cases, functionalities, and actions. They ensure alignment and understanding among stakeholders but often contain technical jargon that hampers comprehension and communication [1]. Extracting key information (such as entities) in the context of the software engineering domain can be beneficial to the software development team [2], [3]. Named Entity Recognition (NER) [4], an NLP technique, addresses this by extracting entities like actors, actions, operators, resources, events, GUI elements, hardware, and APIs from SRS documents [5], [6]. In the context of SRS documents, NER serves multiple purposes and brings various benefits. It helps stakeholders elicit requirements, assess the impact of changes on related components, and facilitate effective change management [7]. Moreover, NER assists in ensuring compliance with legal and regulatory requirements, as well as industry standards and guidelines [3]. Additionally, in source code analysis, NER helps to identify important entities within software code [8] and error messages, thereby aiding in identifying entities related to error messages, stack traces, and code snippets, thereby expediting the debugging process [9].

Previous research in named entity recognition within the software engineering domain highlighted several challenges that need to be addressed to build an effective NER mechanism for the software requirement-specific domain [5][6]. These challenges include the lack of annotated datasets for training and testing NER models. Besides, the deficiency of general NER models in covering domain-specific entities, the requirement for high precision and recall in entity extraction to avoid incorrect software development decisions calls for further research. Also the need for continual updating of the NER system for new entities and emerging terminologies in the software requirements domain is felt.

In this paper, we propose two new approaches for named entity recognition in zero-shot settings. These methods are particularly useful in domains where labeled data is in scarce or non-existent. They involve configuring pre-trained language models (PLMs) for specific downstream tasks and are designed to operate without any labeled training datasets. The main contribution of the paper can be summarized as follows.

- A template-based zero-shot learning mechanism that uses the prompt engineering approach [10] to recognize entities. This mechanism has achieved 93% accuracy with respect to a manually created software engineering named entity dataset derived from the PROMISE repository[1].
- A question-answering-based zero-shot learning mechanism that uses question-answer templates to recognize entities. This method yields 98% accuracy with respect to the same dataset.

Several studies (e.g. [5], [11]) have noted the absence of annotated NER datasets in the requirements engineering domain for the purpose of training a supervised model or evaluating a zero-shot model. As an indirect outcome of this research work, we have created two datasets for the requirements engineering community and made them available publicly [2],[3]. These are as follows:

1) An annotated dataset for training and evaluation of NER models
2) A Natural Language Inference (NLI) dataset tailored for the requirements engineering domain that can be used for transforming large language models into zero-shot classifiers.

[1]http://promise.site.uottawa.ca/SERepository/datasets-page.html
[2]https://github.com/svk-cu-nlp/zsl-ner
[3]https://zenodo.org/record/8020689

2332-6441/23/$31.00 ©2023 IEEE
DOI 10.1109/RE57278.2023.00019

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

By using these two datasets, we were able to produce an experimental evaluation of our proposed models for named entity recognition in the requirements engineering domain and to compare them against existing state-of-the-art approaches. In both cases, our zero-shot NER mechanisms achieve better accuracy than existing state-of-the-art solutions.

The rest of the paper is structured as follows: In Section II, we provide a background on zero-shot learning using prompt engineering approach and language models. In Section III, we present our first framework for zero-shot NER using a prompt engineering approach. In Section IV, another approach for NER using Question-Answering task has been presented. In Section V, we mention some threats to the validity of the proposed approaches. In Section VI, we present the existing state-of-the-art for named entity recognition in the software engineering domain. Finally, Section VII concludes.

II. Preliminaries

This section provides essential NLP concepts for developing zero-shot named entity recognition models.

A. Zero-shot Learning and Few-shot Learning

Large-scale annotated dataset availability is necessary for deep learning techniques. A significant challenge in performing downstream tasks arises when publicly available annotated datasets are scarce in specific fields. For example, requirements engineering domain lacks with named entity dataset upon which a deep learning model can be trained to recognize different entities from the natural language requirements. In this scenario zero-shot and few-shot learning come to the rescue [12], [13]. According to [14], an approach to scaling up the recognition task by developing such models capable of recognizing unseen classes without any prior training is called zero-shot learning. On the other hand, the goal of few-shot learning is to recognize novel classes without performing rigorous training rather than introducing adaptability of the deep learning model using just a few illustrative examples [15].

Table I: Notations and steps involved in prompt learning.

Name	Notation	Example	Explanation
Input	$\boldsymbol{x}$	On clicking privacy policy link, a window must appear.	Input text
Output	$\boldsymbol{y}$	GUI	
Possible Labels	Output label $\boldsymbol{z}$	{Event, GUI, Hardware, Platform, Organization, ...]	Set of labels from which the output will be decided.
Prompt function	$f_{prompt}(x)$	[X1]. The entity type of the word [X2] is [Z]	A function creates a template from input and adds a slot Z that will be filled later.
Prompt	$\acute{x}$	On clicking privacy policy link, a window must appear. The entity type of the word window is [Z]	The modified text where [X] is filled but [Z] is not.
Filled Prompt	$f_{fill}(\acute{x}, z^*)$	On clicking privacy policy link, a window must appear. The entity type of the word window is Event	[Z] can be filled with any answer.
Answer Prompt	$f_{fill}(\acute{x}, z)$	On clicking privacy policy link, a window must appear. The entity type of the word window is GUI	[Z] is filled with true answer.

Table II: Types of Prompt Template

Prompt Template	Type
[X]. The review is about [Z]	Prefix Prompt
[X] : The review specifies [Z] type of category.	Cloze Prompt

B. Prompt Learning and Template Design

One of the latest approaches to achieve zero-shot learning is prompt-based learning which involves developing templates for specific NLP tasks. The process of building prompting functions or templates depending upon the input text data and the specific task is called prompt engineering. Traditional supervised learning trains a model to take an input $\boldsymbol{x}$ and predict an output $\boldsymbol{y}$ denoted as $\boldsymbol{P(y \mid x)}$. On the other hand, prompt-based learning is based on language models that model the probability of text directly [12]. The template is developed on top of the original text $\boldsymbol{x}$ and usually, the template has some unfilled slots. The language model performs the prediction tasks based on the template. It predicts the text for the unfilled slots and generates the string $\hat{\boldsymbol{x}}$. The final output $\boldsymbol{y}$ is then derived from $\hat{\boldsymbol{x}}$.

The *prompting function* $f_{prompt}(x)$ is applied on input $\boldsymbol{x}$ and transform the input into $\acute{x}$. Thus $\acute{x} = f_{prompt}(x)$. At first, we define a template, that includes two slots - an input slot [X] for input $\boldsymbol{x}$ and an answer slot [Z]. The slot [X] is filled by the input text $\boldsymbol{x}$. The answer slot can be filled with any labels as the intermediate answer text. Later, the highest scored predicted label will be mapped into $\boldsymbol{y}$. In Table I, we present notations and steps involved in prompt learning with corresponding examples. There are two types of prompt possible in prompt engineering. In Table II, we show examples for each of the categories of prompt template. In *Prefix prompt*, the tokens predicted by a language model fill the subsequent masks. In case of *Cloze prompt*, the slots are in somewhere middle of the string and it can be viewed as fill in the blank query.

C. Natural Language Inference and Text Entailment

A directional relationship between text fragments is known as Textual Entailment (TE) [16]. When the truth of one text fragment follows from another text, the relationship is valid. The entailing and entailed texts are referred to as text and hypothesis, respectively, in the TE framework. This imitates how humans understand the meaning of an aspect and derive hypothesis from it. For example, if we consider the statement (S)- "*on clicking privacy policy, a window will appear*" and possible candidates for entity type of *window* {*Event, Webpage*}, we humans generally, construct a hypothesis by constructing a text like "the entity is about ___" and by filling the mask or blank space using candidates e.g., "Event" or "Webpage". Next, we try to evaluate whether this hypothesis is true for the given the statement S. It has recently been reported that Recognizing Textual Entailment (RTE) be used as a general job to address the core semantic inference requirements of numerous natural language processing applications [17]. In this context, it is worth mentioning that two popular datasets

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

MNLI [16] and RTE [18] are available to train deep learning models on top of it and leverage the knowledge of entailment relationship in zero-shot or few-shot classification tasks.

D. Pre-trained Language Model

Pre-trained language models (LMs) have revolutionized NLP tasks, achieving state-of-the-art performance through pre-training and fine-tuning. These models optimize the probability of text using autoregressive prediction, typically from left to right. The suitability of pre-trained LMs for various prompting tasks depends on their training objective, with left-to-right autoregressive LMs performing well for prefix prompts and reconstruction aims better for cloze prompts. For text classification, training a language model on a natural language inference dataset like MNLI enables it to function as a zero-shot text classifier, providing an effective solution for the text entailment problem.

1) Notable Language Models Used in this Research:
GPT-3: GPT-3 (Generative Pretrained Transformer 3) [19] is a large language model developed by OpenAI that has earned significant attention in the natural language processing community. It is trained on a massive amount of data and is able to generate human-like text that is coherent and diverse. One of the key innovations of GPT-3 is its use of a hierarchy of transformer models, with the smallest model (GPT-3 "Small") having 125 million parameters and the largest model (GPT-3 "175B") having 175 billion parameters. This allows GPT-3 to capture a vast amount of knowledge and information about the structure and use of language, but also makes it computationally expensive to run and fine-tune.

T5: The Text-To-Text Transfer Transformer language model T5 [20] is a state-of-the-art natural language processing (NLP) model developed by Google Research. It was introduced in 2020 and has since been applied to a variety of NLP tasks, including translation, summarization, and language modeling. One of the key innovations of T5 is its use of a unified input-output representation for all tasks. This representation consists of a pair of special tokens, called "task tokens," that are added to the beginning of the input and output sequences to specify the task that the model is being asked to perform. For example, the task tokens "translate English to French: " and "translation: " would be added to the beginning of an input and output sequence, respectively, to indicate that the model should perform translation from English to French. This allows T5 to be fine-tuned for various tasks using the same model architecture and training data, without the need for task-specific modifications or training regimes.

III. Prompt Based Zero-shot Named Entity Recognition

In this section, we present a zero-shot named entity recognition mechanism capable of identifying entities without the need for a training dataset. The mechanism operates within the framework depicted in Figure 1. Instead of training on data, it employs prompt templates and utilizes the language model's knowledge to recognize entities within the given context. Prompt template generation is crucial for establishing the relation between the context and template. In the final stage, the language model predicts the entity type of the highlighted entity in the generated template. The recognized software engineering entities are highlighted in Table III. The selected entities are also aligned with state-of-the-art entity taxonomy [5], [6].

Table III: Entity types used for the annotation of the dataset

Entity	Description	Example
O	Non-Entity Tokens	Tokens do not belong to entity set
VART	Virtual Artifacts	Case History, Record, Account
SSW	Standard Software Components	Anti-virus, URL, PDF, JPG, Database
VER	Version	chrome 91.67, Eclipse Photon
GUI	Components of Graphical User Interface	Widget, Page, Applet, Wizard
HW	Hardware Components	Printer, Display, Projection Screen
PLAT	Application or Software Platforms	Browser, CRM, ERP
OS	Operating System	Windows, Ubuntu, Mac
EVE	Event or Activity	Screening, Beta-Testing,
TIM	Time Indicator	Minute, Second, Hour, Date, Day, Month, Year
ORG	Organization	Apple, Microsoft, IBM
ACTOR	Entity which has a role	Admin, Developer, Client, System
LANG	Programming Language	C, C++, Java, Python

A. The Framework

At its core, this framework employs prompt engineering and adept language model training to construct a seamless zero-shot learning model for named entity recognition, emulating the human-like understanding. The template is designed in such a way that it can utilize the maximum capabilities of language models in order to understand a sentence and be capable of finding relevancy with other sentences. The entire configuration of templates and the language model results in the zero-shot models for the intended NLP task. Before going into a detailed explanation of the modules of the framework, we highlight a crucial aspect that is the main foundation

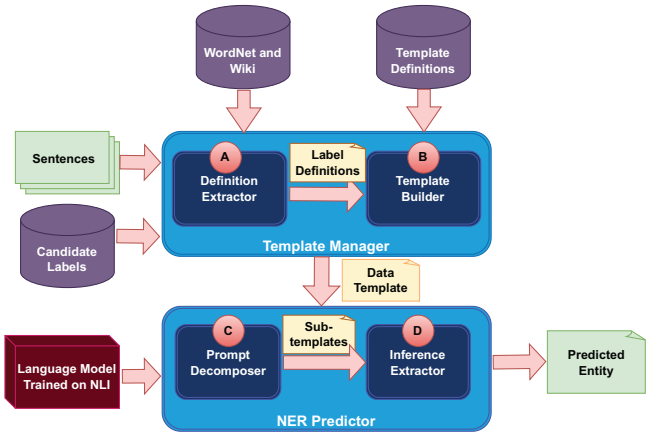

Figure 1: Architecture of Zero-shot Named Entity Recognition

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

of the framework, namely the *Textual Entailment for Zero-shot NER*. The text entailment approach tries to mimic the humans' perspective on the text classification problem. The textual entailment enables us to design a generalization of the classifier. The framework elegantly tackles the NER problem by harnessing text entailment, effectively capturing semantic coherence and compatibility between text and labels. It adeptly determines if a given text implies or contradicts a specific label, even without prior exposure to corresponding training examples.

The framework comprises several modules which in turn help to build the model to recognize named entities. We elaborate on all the technicalities of each of the modules in the following subsections.

1) Template Manager: Template Manager is the first module of the framework. It comprises two submodules - *Definition Extractor* and *Template Builder*. The main objective of this module is to build different data templates (i.e. the input slots are filled appropriately) based on the defined templates so that the language model exhibits different behaviors for different contexts created by the template. The inputs of the module and the roles of the sub-modules are as follows.

(i) ***Inputs:***

a) *Sentences:* The sentences containing entities specified within it. For example - we specify two entities in the sentence "*The dashboard (B-ENT) must contain navigation (B-ENT) menu (I-ENT)*". We follow the BIO[4] format and *ENT* tag to annotate the entities that are to be recognized by the framework.

b) *Candidate Labels:* Candidate labels are the set of entities that are to be considered in order to recognize the tagged entities in the input sentences. The recognized entity type must be mapped to one of the classes from the candidate labels.

c) *WordNet and Wikipedia Definitions:* We leverage the extensive vocabulary provided by WordNet [21] and Wikipedia to obtain the definitions of candidate labels and specific entities highlighted in the input sentences. The English WordNet 2019 [21] contains 117,791 words and their corresponding lemma, synset, and examples.

d) *Template Definitions:* The template definitions are a crucial aspect of this framework. As shown in Table IV, a variety of templates have been prepared for use within the framework. These templates are kept readily available to ensure that the subsequent modules can easily access them whenever necessary.

(ii) ***Definition Extractor:*** The Definition Extractor module (marked as A in Fig.1) extracts definitions of specific words from both WordNet and Wikipedia, and stores them in separate lists. It then provides these definitions, along with candidate labels, as output for the highlighted words in the input sentences.

[4]In software requirement context, BIO can be interpreted as *Beginning*, *Inside* or *Outside* of the entity tag.

Table IV: Different Templates Used for Named Entity Recognition Task

Prompt Template	Type of Prompt	Remarks
[X1]. The entity [X2] is of type [Z]	Prefix Prompt	First Simple Prompt.
[X1]. The type of entity of [X2] is [Z]	Prefix Prompt	Second Simple Prompt.
In sentence [X1], the entity [X2] means [X_D]. It represents [Z]	Prefix Prompt	Prompt involving definition of the entity.
In sentence [X1], the entity [X2] means [X_D]. It is related to [Z] i.e. [Z_D].	Prefix Prompt	Prompt involving definitions of both entity and the candidates.
X1]. The entity [X2] is related to [Z] i.e. [Z_D].	Prefix Prompt	Prompt involving definition of the candidates only.

(iii) ***Template Builder:*** The main responsibility of the *Template Builder* (marked as B in Fig.1) is to accept the sentences and create the data template by filling the input slots. Different template definitions must be present beforehand to make this module operational.

Table IV shows different templates that are used in this framework. The first two templates have two input slots. *X1* will be filled with the input sentence and *X2* will be filled with the entity tagged with *ENT* token. We add an additional input slot i.e. the definition of the entity (X_D) in the third template. The fourth template contains two additional slots. The X_D and Z_D slots represent the entity definition and the candidate label definition, respectively. It provides the word (Entity or the Candidate label) to the *Definition Extractor* module and receives the definitions in order to fill the input slots X_D and Z_D. The idea behind creating a wide range of contexts using these four templates is that the language model might get a chance to respond accordingly and come up with a more accurate prediction. Thus, using these templates, we monitor the framework's activity and evaluate its effectiveness.

In this stage, we have created the necessary background in order to perform the text entailment task. The second module is responsible for the mechanism of creating hypotheses from the candidate labels.

2) NER Predictor: The *NER Predictor* module is responsible for predicting named entities based on the data template generated by the *Template Manager*. However, before delving into the specifics, it's worth mentioning the primary objective that we aim to achieve through this module.

Converting Candidate Labels to Hypothesis: As discussed in Section II, textual entailment problem can be viewed as a relation between two statements - *Text* and *Hypothesis*. The relation is recognized as *Entailment, Contradiction* or *Neutral*. In this scenario, the text is already available, i.e. the given entity with a given context as a sentence. In order to create the hypothesis statement, we follow two methods - one is by using the label's name, and the other is by using the label's definition. In order to provide a better representation, we add a piece of text along with the label and its definition in the hypothesis sentence. In Table IV, the first sentence of the template acts like the *Text* and the second sentence acts as the hypothesis. In summary, for each template, the output slot(s) are filled by the number of candidate labels and with

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

corresponding definitions (if required). This creates a unique instance for that particular template. The number of instances of each template will be equal to the number of candidate labels.

The *NER Predictor* module involves two sub-modules - *Prompt Decomposer* and *Extract Inference* in order to predict the category of the intended entity. The module utilizes the mask word prediction and natural language inference capabilities of the language model to recognize the sentence pair (Text and Hypothesis) as the entailment problem and derive the inference. Inputs and the role of each component involved in this module is elaborated in the following:

(i) ***Inputs:***

a) *Data Template:* Each Data template, generated by the *Template Manager* is the text consisting of an input sentence and output slots filled with one of the candidate labels and its definition. In summary, each data template consists of two sentences - the input sentence and the appended text composed of filled output slots.

b) *Language Model Trained on NLI:* The *NFR Predictor* module involves a language model that is trained on NLI dataset to derive the inference between two sentences in the data template. In this research, we used the T-5 model and trained it on NLI dataset to make it compatible to encounter the text entailment problem.

(ii) ***Prompt Decomposer:*** The data templates generated by the *Template Manager* are accepted by the *Prompt Decomposer* module (marked as C in Fig.1). It may often be the case that a sentence can contain more than one entity, and we are interested in recognizing all of them. In such cases, the prompt templates become much more complex, as they involve the explicit definitions of the entities and labels in some of the templates. This module decomposes the templates into multiple sub-templates and prepends the input text as well to keep the context information available to the language model. Figure 2 shows one example of the template decomposition.

(iii) ***Inference Extractor:*** The *Prompt Decomposer* then passes the templates or sub-templates to the *Inference Extractor* module (marked as D in Fig.1) where it utilizes the pre-trained language model in order to get the inference. The language model must be pre-trained in such a way that it can recognize the NER task as a text entailment task. The pre-trained language model recognizes the *Text* and *Hypothesis* sentence pairs from the data templates and extracts the inference based on the knowledge the language model has. We used BERT [22], Roberta [23], BART [24], and T5 [20] language models to train on the MNLI [16] dataset. Based on the text templates and sub-templates provided by the *Prompt Decomposer* module, the language model predicts the inference with corresponding scores for each instance of a given template. The instance with the highest score is determined to be the most appropriate instance for a particular template. This in turn results in the prediction of the most appropriate label for the given entity specified in the template.

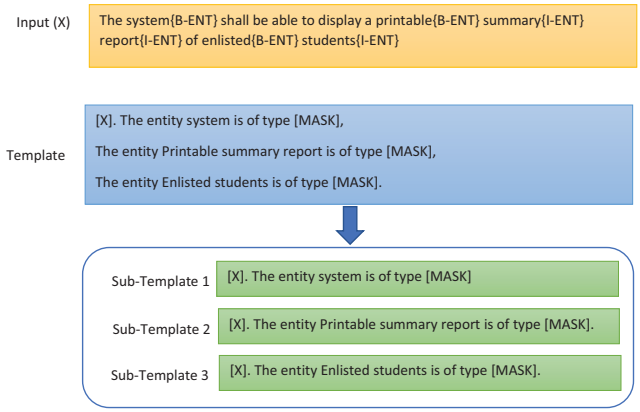

Figure 2: Decomposition of Template into Sub-templates

B. Experimental Evaluation

1) NLI Datasets in Software Engineering domain: Although there are several NLI datasets available for general English, we need to create a domain-specific NLI dataset specifically for the software engineering domain. This is important because the language used in the software engineering domain may differ from generic English. We consider different texts from the software engineering domain and manually curated the entailment relationships among different sentences. The source of the sentence pairs includes PROMISE dataset, PURE dataset[5], user guides of different softwares, articles on operating systems (e.g. official documentation of Windows, Mac), databases (e.g. official documentation of MongoDB, Oracle), cyber security (MITRE documentation) and software product descriptions including AWS documentation. Our created text entailment dataset contains 10k sentence pairs manually labeled for balanced classification with the labels *Entailment*, *Contradiction*, and *Neutral*. There are 3.400 sentence pairs labeled as *Entailment*, 3.300 labeled as *Contradiction*, and 3.300 labeled as *Neutral*. During the creation of the dataset, a total of 5 annotators, including 2 authors of the paper, actively participated in the annotation process. Each annotator was assigned the task of generating 2000 sentence pairs which were then shared among the other annotators of the team for annotation. Validation involved all annotators and authors, ensuring quality and accuracy. The majority voting consensus finalized the labels for each sentence pair.

2) Test Dataset Generation: The test dataset with respect to NER task has been developed using the sentences taken from the PROMISE dataset. The dataset consists of 600 sentences tagged with an average of 5 named entities per sentence.

[5]https://zenodo.org/record/1414117#.ZAr4u3bMK3A

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

Table V: Example sentence annotated with entity labels in BIO format.

System	shall	have	MDI	form	that	allows	user	to	view	graph	in	the	Screen
B-PLAT	O	O	B-GUI	I-GUI	O	O	B-ACTOR	O	O	B-VART	O	O	B-HW

Table VI: Running Example of Prompt Based NER

Sentence	Candidate Label	Data Template	Entailment Score	Predicted Answer
The firewall/I-ENT fails to protect the Bank server	Hardware	The firewall fails to protect the Bank server. The entity firewall is of type Hardware	0.6	Software
The firewall/I-ENT fails to protect the Bank server	Software	The firewall fails to protect the Bank server. The entity firewall is of type Software	0.8	
The firewall/I-ENT fails to protect the Bank server	GUI	The firewall fails to protect the Bank server. The entity firewall is of type GUI	0.4	

entities. In Table III, we present a diverse set with thirteen (13) entity labels - twelve (12) entity categories and one (1) non-entity label - which covers a wide range of entities present in the software engineering domain. The creation of the test dataset involves similar participation as in dataset creation. To ensure the quality and accuracy of the dataset, manual checking of the tagged entities has been conducted involving all annotators and authors. No lemmatization or stemming technique has been applied as we opted to keep the semantic structure of words similar to the documents prepared in the software development processes. In order to create the annotations, we apply the widely accepted BIO representation scheme. Table V shows an example of a natural language requirements sentence from the dataset which has been annotated in the BIO format. Finally, we examine the annotations manually in order to ensure high-quality data.

3) Template Creation: We use the OpenPrompt [10] library that provides required libraries and constructs to build the prompt engineering pipeline. The template creation constructs of the OpenPrompt library help us to define five (5) different templates as discussed in Section III-A1 (Table IV). There are also provisions to provide the required texts for filling up the input slots in the template. The definitions for the entities and the labels are extracted from Wikipedia by web crawling. In order to avoid ambiguities among the words, we append text such as *"software engineering"* with the searched text (entity).

4) Selection of Language Model: The next step is to select a language model for predicting the token for the output slot of the templates. Given that T5 currently holds the highest level of performance for the task of natural language inference and surpasses previous state-of-the-art models BART and RoBERTa [20], we utilize three transformer-based language models, including *RoBERTa, BART*, and *T5*, to train them on a natural language inference dataset. We combine the MNLI [16] dataset with a specialized NLI dataset focused on the software engineering domain to gain a comprehensive understanding of natural language entailment relationships while also incorporating domain-specific knowledge in the form of software-related terminology. The hyper-parameters for the training is summarized in Table VII.

Table VII: Hyperparameters for training T5 model on NLI dataset

Hyperparameters	Value
learning_rate	3e-4
Optimizer	AdamW
Loss function	CrossEntropyLoss
max_seq_length	256
batch_size	16

We also evaluate their performance with respect to the natural language inference task on the 20% split of the same dataset. We found that the T5 model outperforms the BART-large and RoBERTa models. The T5 model achieves an F1-score of 90.3% and the large variant of the BART model achieves an F1-score of 86.6%, whereas RoBERTa model achieves an F1-score of 85.25%. It is noteworthy that while the highly advanced GPT-3 model could be utilized in this scenario, the process of fine-tuning the GPT-3 model on the NLI dataset incurs a significant cost. Thus, we decide to use the T5 model for answer prediction from the templates.

5) Evaluation and Result: We evaluate our framework on the created dataset discussed in Section III-B2. Table VIII presents the performance of our framework for different templates. The first two templates are identical and perform quite similarly with the accuracy of 92.01% and 91.10% respectively. On the other hand, the third and fourth templates with embedded definitions for the entities do not work well. With respect to the third template, the framework achieves 84.33% of accuracy, and for the fourth template, it achieves up to 78.11% of accuracy. The reason behind the significant performance drop for these two templates is that the input text and the definitions of the entities are exaggerating and diversifying the entire context. Instead of involving the definitions for entities in the input text, the final template involves only the label and its definition, which significantly enhances

Table VIII: Evaluation of different templates on software requirements NER dataset

Prompt Template	Precision	Recall	F1-Score	Accuracy
[X1]. The entity [X2] is of type [Z]	92.22%	93.33%	92.77%	92.01%
[X1]. The type of entity of [X2] is [Z]	91.39%	92.49%	91.94%	91.10%
In sentence [X1], the entity [X2] means [X3]. It represents [Z]	87.18%	86.20%	86.69%	84.33%
In sentence [X1], the entity [X2] means [X_D]. It is related to [Z] i.e. [Z_D].	81.61%	81.14%	81.38%	78.11%
X1]. The entity [X2] is related to [Z] i.e. [Z_D].	95.18%	94.13%	94.65%	93.68%

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

the performance with respect to the previous two templates and performs best among the five templates. The framework achieves 93% of accuracy and 94.65% f1-score with the fifth template.

IV. Question-Answering Based Zero-shot Named Entity Recognition

The former proposed approach transforms the NER problem into a text entailment task, employing NLI models. However, it relies on manual prompt engineering for pre-defined prompt formulation. To address this limitation, a zero-shot NER model is introduced in this section. This model automates prompt formulation by treating NER as an extractive Question-Answering (EQA) task. In EQA, the goal is to extract the answer span from a context given a question. By asking specific questions to the EQA model, entities can be recognized in the context. For instance, in the context "The firewall fails to protect the Bank Server," asking the question *What is the software?* would extract *Firewall* as the answer. Similarly, asking *Which is the organization?* would yield *Bank* as the answer. These examples illustrate the working principle of the proposed zero-shot model.

A. The Framework

Figure 3 demonstrates the framework of the proposed Question-Answering(QA) based named entity recognition mechanism. We elaborate on all the technicalities of each of the modules in the following subsections.

1) Question Prompt Generator: The required prompt to make the EQA model operational is the question itself. The generation of questions depending upon the set of candidate labels is the main challenge. The generic template used in this work is described as follows.

$$f_{prompt}(x)\text{: [MASK] is the [X]?}$$

The template contains a `MASK` token and an input slot. In order to resolve the `MASK` token in the prompt, a generic language model is definitely of need. In this work, we use BERT to predict the `MASK` token to formulate the prompt.

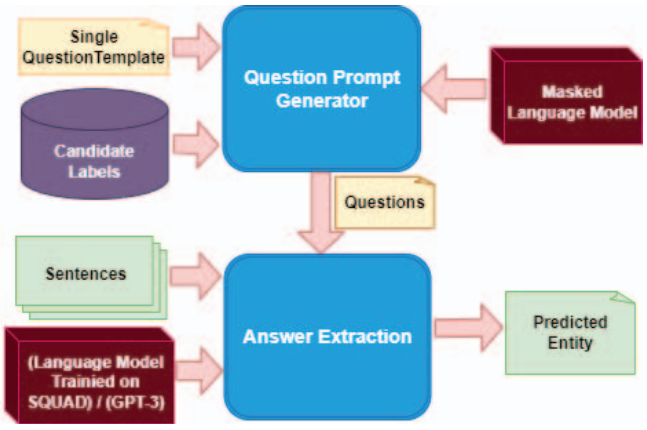

Figure 3: Framework of QA-based Zero-shot NER

The input slot is filled by the candidate labels which in turn generates different instances of the filled prompt. Examples of filled prompts are presented as follows.

$$f_{fill}(x_1') \text{ : [What] is the Software?}$$
$$f_{fill}(x_1') \text{ : [Which] is the Organization?}$$

In order to predict the `MASK` token we take top k = 5 predicted words and formulate the questions. Irrelevant prediction of words is inevitable and it is also evident that omitting of the question mark sign (?) from the template results most diverse set of predictions. We keep the question mark attached to the prompt in order to get a consistent prediction.

2) Answer Extraction: We build the answer extraction models by training T5 (present state-of-the-art model for SQUAD) and RoBERTa (previous state-of-the-art) [20] models on SQUAD 2.0 [25] question-answering dataset. SQUAD 2.0 is an improved version of SQUAD 1.1 [26], containing additional 50,000 unanswerable questions crafted to resemble answerable ones. We discard the *unanswerable* label extracted from the context. In the next stage, we allow the model to extract answers from the context based on the questions generated by the prompts. We consider top $k = 5$ predicted words for the `MASK` token. This results in 5 different question prompts for each class of the candidate labels. The answer to the different question prompts may result in different predictions. In order to select the proper question to get an accurate final prediction, we use the weighted majority voting scheme [27]. In the weighted majority voting, we assign weights to each of the generated questions in descending order i.e. question with the highest scoring `MASK` word will get the highest weight, and the question with the lowest scored `MASK` word will get the lowest weight. The weight ranges from 0 to 1.

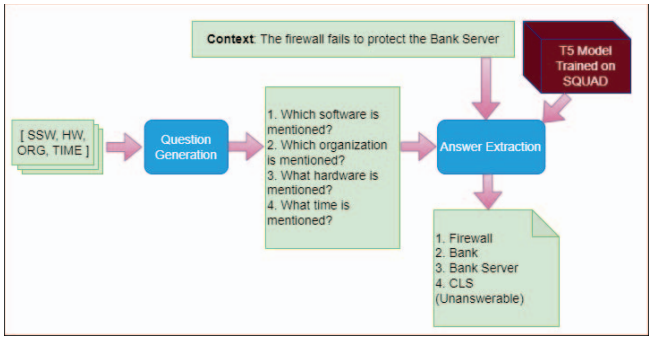

Figure 4: Question-Answering based Zero-shot NER in Action

Let us summarize the entire process with the help of Figure 4. The first module accepts the candidate labels and generates questions by filling the `MASK` token with the appropriate WH-words. The second module (*Answer Extraction*) accepts the given context. It utilizes the pre-trained language model trained on SQUAD 2.0 dataset in order to generate the answers for each of the questions. Table IX presents some examples of QA-based NER activity.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

Table IX: Running Example of Question-Answering based NER

Sentence	X	Predicted Prompt	Score	Predicted Answer
The firewall fails to protect the Bank server	Software	What software is mentioned in the sentence?	0.82	firewall
		Which software is mentioned in the sentence?	0.75	
		Where software is mentioned in the sentence?	0.42	
		How software is mentioned in the sentence?	0.04	
The firewall fails to protect the Bank server	Hardware	What Hardware is mentioned in the sentence?	0.79	Bank Server
		Which Hardware is mentioned in the sentence?	0.85	
		Where Hardware is mentioned in the sentence?	0.51	
		How Hardware is mentioned in the sentence?	0.03	
The firewall fails to protect the Bank server	Organization	What Organization is mentioned in the sentence?	0.49	Bank
		Which Organization is mentioned in the sentence?	0.84	
		Where Organization is mentioned in the sentence?	0.72	
		How Organization is mentioned in the sentence?	0.43	
The firewall fails to protect the Bank server	Time	What Time is mentioned in the sentence?	0.82	CLS
		Which Time is mentioned in the sentence?	0.41	
		Where Time is mentioned in the sentence?	0.22	
		How Time is mentioned in the sentence?	0.46	

B. Experimental Evaluation

1) Environment for Question Generation Task: In this technique, we transform the NER problem into a QA task where the answer is extracted from the context and the entity is classified into candidate categories. By leveraging the language model's power, we generate relevant questions based on the candidate labels. The prompt includes a `MASK` token, and using BERT, we predict the appropriate word for the `MASK` token based on the candidate label. The automatic question generation process produces the same number of questions as the candidate labels. The most frequent and meaningful predicted words for the `MASK` token are "Which" and "What" in our software engineering domain. BERT, being a masked language model, is advantageous for training as it maintains contextual representations for all input tokens. Additionally, BERT's bidirectional nature allows it to predict `MASK` words anywhere in the text. Considering GPT-3 or T-5 models would be costlier and unnecessary, as BERT achieves over 90% F1-Score in predicting `MASK` words [22], [28].

2) Environment for Answer Extraction: In order to extract answers from the context, the language models RoBERTa and T5 are fine-tuned using the SQUAD 2.0 question-answering dataset. The fine-tuning process follows the parameters mentioned in the original T5 [20] and RoBERTa [23] papers. We choose these two models because the current state-of-the-art T5 model achieves 90.7% of F1-Score for SQUAD2.0 dataset whereas the RoBERTa model archives 88.6% of F1-score for the same. In addition to these language models, we also utilize the Text-DaVinci-003 variant of GPT-3, capable of handling complex instructions [29]. While GPT-3 achieves an F1-score of 85% [19] on zero-shot question-answering tasks, it falls slightly behind the current state-of-the-art fine-tuned models. However, Text-DaVinci-003, with its comprehensive knowledge base and advanced capabilities, eliminates the need for additional fine-tuning on the SQUAD 2.0 dataset. We compare the performance of GPT-3 with the aforementioned approaches in the following section.

Table X: Evaluation of Question-Answering Based Zero-shot NER for Different Language Models

Answer Extraction Language Model	**Precision**	**Recall**	**F1-Score**	**Accuracy**
RoBERTa trained on SQUAD 2.0	86.44%	87.24%	86.66%	86.26%
T5 trained on SQUAD 2.0	88.65%	87.48%	88.25%	88.65%
GPT 3 text-davinci-003 model without training	98.25%	97.87%	98.85%	98.65%

3) Result and Analysis: With above mentioned environmental settings, we conduct the experiment to recognize entities in the software engineering domain. In Section III-B2 we have already described our test dataset for NER task in software engineering domain. We have evaluated our approach using three language models on the test dataset. Table X, presents a comparative analysis of the different models for recognizing the entities based on the question-answering task. It is notable from the table that, we achieve a significant performance of 98.85% of F1-Score with GPT-3 Text-DaVinci-003 model. GPT-3 Text-DaVinci-003 model [19] outperforms the other two fine-tuned models with a significant gap and achieves state-of-the-art performance with respect to NER tasks in the software engineering domain. We conducted an experiment to test the limits of the GPT-3 model's named entity recognition abilities in the software engineering domain. Instead of providing it with pre-generated questions, we simply gave it a question prompt and asked it to identify various entities related to software engineering. While the GPT-3 model demonstrated strong performance in this task,

Table XI: Some of the incorrect results of NER by GPT-3 model without Question-Answering task in pipeline

Sentences	**Incorrect Entity Recognition in Software Engineering Domain**
The application shall match the color of the schema set forth by Department of Homeland Security	Color: Visual Property
	Schema: Psychological Behavior
If projected the data must be readable from a viewing distance	Viewing Distance: Measure
If projected the data must be understandable.	Understandable: Understandability
The Disputes application must conform to the legal requirements as specified by the Merchant Operating Regulations.	Merchant Operating Regulations: Organization
An amount of $10 will be credited to the account	$10: Currency

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

Table XII: Examples of Zero-shot NER on Texts from Different Domain

Approach	Predicted Entity Tagged Sentence	Additional Set of Specified Entity
QA based NER Model	Cloudflare's\B-ORG firewall\B-PROD includes bot\B-TECH management\I-TECH to protect against DDoS\B-ATTACK attacks\I-ATTACK.	{Technology, Product, Attack}
Prompt-based NER Model	Cloudflare's\B-ORG firewall\B-TECH includes bot\B-TECH management\I-TECH to protect against DDoS\B-ATTACK attacks\I-ATTACK.	{Technology, Product, Attack}
QA based NER Model	Cloudflare's\B-PROD firewall\I-PROD can block certain IP\B-VART addresses\I-VART from accessing your website\B-PLAT, providing an additional layer of security.	{Technology, Product} and Omitted {Organization}
Prompt-based NER Model	Cloudflare's\B-PROD firewall\I-PROD can block certain IP\B-VART addresses\I-VART from accessing your website\B-PLAT, providing an additional layer of security\B-TECH.	{Technology, Product} and Omitted {Organization}
QA based NER Model	We have implemented two-factor\B-TECH authentication\I-TECH to increase the security of our users'\B-VART accounts\I-VART.	{Technology, Product, Attack}
Prompt-based NER Model	We have implemented two-factor\B-TECH authentication\I-TECH to increase the security of our users'\B-ACTOR accounts\I-VART.	{Technology, Product, Attack}
QA based NER Model	One of the major updates in Chrome\B-PLAT 108.0.5359.125\B-VER is the ability to run web\B-SSW apps\I-SSW in a more secure\B-TECH sandbox\I-TEC environment\I-TECH.	{Technology, Product}
Prompt-based NER Model	One of the major updates in Chrome\B-PLAT 108.0.5359.125\B-VER is the ability to run web\B-SSW apps\I-SSW in a more secure sandbox\B-TECH environment\I-TECH.	{Technology, Product}
QA based NER Model	The virtual\B-FEAT background\I-FEAT on Zoom\B-ORG is a complete disaster. It never works properly and just ends during meetings\B-EVE.	{Technology, Feature, Product}
Prompt-based NER Model	The virtual\B-FEAT background\I-FEAT on Zoom\B-PLAT is a complete disaster. It never works properly and just ends up during meetings.	{Technology, Feature, Product}

we found that it tended to exaggerate its existing knowledge and recognize more general entities such as location and currency, rather than specifically focusing on entities related to software engineering. Instead of recognizing only the entities, it also recognizes different concepts like functional and non-functional requirements. Table XI shows such examples of recognized elements that are not relevant to the software engineering domain. However, to ensure more accurate and targeted results, we have limited the application of the GPT-3 model to only recognizing a predefined set of classes of named entities within a given context.

In addition to the software engineering domain, our experiments extended to recognizing entities in different domains, including information security and App reviews. The test sentences were sourced from software release logs (e.g., Cloudflare, Chrome, Windows) and App reviews from Google Playstore. We conducted the experiment on 500 sentences, reaffirming the capability of our models to perform NER without prior learning. Table XII showcases examples of extracted entities from these test sentences in the zero-shot setting.

V. Threats to Validity

The main threats to the validity of this research may be summarized as follows.

- **Construct Validity:** The NER models uses the prompt based approach and we evaluate the performance of the model involving different prompts. The lack of a systematic analysis of the impact of the construction of the prompt templates may be seen as a threat to construct validity.
- **Internal Validity:** We relied on a manually created named entity dataset for evaluating the proposed methods. The dataset's characteristics and diversity might not fully represent the software engineering domain, impacting the generalizability of the results. The subjectivity of the annotation process for the named entity dataset poses a potential threat to internal validity as it may introduce the possibility of biases and inconsistencies in the labeling of entities. On the other hand, utilization of sentence pairs from the PROMISE dataset to create the Natural Language Inference (NLI) dataset may influence the evaluation results.
- **External Validity:** The performance of the NER models can be influenced by the choice of pre-trained language models (PLMs) and the settings of hyperparameters. However, it is important to consider that the generalizability of the proposed methods to different PLMs or variations in hyperparameters might be limited, which can impact the external validity of the research. This research focuses on NER in the software engineering domain and the performance of the proposed models in other domains (such as information security, software privacy, and compliance policies) may vary.

VI. Related Work and Comparative Analysis

The SoftNER model [8] is a BERT-based model that has been designed to perform well on StackOverflow data for the identification of code tokens and software-related token labels. DBNER [9] is a deep neural network model that was specifically developed for bug-specific entity recognition. It employs a combination of bidirectional LSTM (BiLSTM) and conditional random field (CRF) models, and incorporates an attention layer to improve the accuracy of the entity recognition process. Another work [3] proposes an automated approach for privacy requirements engineering using named entity recognition (NER) to detect privacy-related entities in user stories. Arora et. al. [2] proposed an automated approach for extracting candidate glossary terms and their related terms from requirements specification documents, which clusters the extracted terms by relevance. The approach also provides mathematical procedure for selecting the number of clusters.

A software-specific NER method (S-NER) is introduced in [6], that identifies the design challenges in creating a NER methodology for social content data. It annotates Stack Overflow posts with five tags, including API, platform, software standard, programming language, and tools, and trained an ML-CRF model with software-specific gazetteers and unsupervised word clusters. In the same line, [30] defined a set of 22 software-specific entities in Stack Overflow posts, including programming languages and software tools, frameworks, and

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

Table XIII: Comparison of state-of-the-art NER approaches with proposed approaches

Approach	Model	Type of Approach	Dataset	Performance Metric	Classify Entities to Unseen Classes?
Tabassum et. al. [8]	BERT	Supervised	StackOverflow (Train + Test)	Precision - 78% Recall - 79% F1 - 79%	No
Zhou et. al. [9]	BiLSTM + CRF	Supervised	Bug reports (Train + Test)	F1 - 91%	No
Ye.et.al. [6]	CRF + Brown Clustering	Semi supervised	StackOverflow (Train + Test)	Precision - 82% Recall - 74% F1 - 78%	No
Reddy et. al. [30]	BiLSTM + CRF	Supervised	StackOverflow (Train + Test)	Precision - 96% Recall - 94% F1 - 95%	No
Malik et. al. [5]	BERT, RoBERTa and ALBERT	Supervised	DOORS, SRE, and RQA (Train + Test)	Precision - 91% Recall - 94% F1 - 92% Accuracy - 95%	Examined Fewshot NER using BART
Herwanto et. al. [3]	RoBERTa, BERT + BiLSTM + CRF	Supervised	Manually labeled user stories (Train + Test)	Data Subject F1 - 91% Processing F1 - 74% Personal Data F1 - 72%	No
Arora et. al. [2]	Text Chunking + Clustering	Heuristics	Manually created ideal set of glossary terms	Precision - 93% Recall - 79% F1 - 85%	Extracts and Clusters Glossary Terms
Tikhomirov et. al. [31]	BERT	Supervised	Sec_col Corpus (Train+ Test)	F-micro - 72% F-macro - 86%	No
Wu et. al. [32]	BiLSTM + CRF	Supervised	Cyber threat intelligence Texts (Train + Test)	Precision - 86% Recall - 84% F1 - 85%	No
Dionísio et. al. [33]	CNN + BiLSTM	Supervised	Manually labeled collected tweets (Train + Test)	TP - 94% TN - 91% F1 - 92%	No
Proposed Approach 1	Prompt Engineering + T5 Model	Zero-shot	Manually created named entity datasets on texts from SRS.(Test)	Precision - 95% Recall - 94% F1 - 94% Accuracy - 93%	Yes
Proposed Approach 2	Transforming NER into QA task + GPT-3	Zero-shot	Manually created named entity datasets on texts from SRS.(Test)	Precision - 98% Recall - 97% F1 - 98% Accuracy - 98%	Yes

protocols. This work revealed that the BiLSTM-CRF model performed better than the ML-CRF model for their NER task. In addition to the above-mentioned works, there have also been studies in the field of cyber security that have focused on entity recognition. RucyBERT [31] is a BERT model trained to tackle textual data specific to the cyber-security domain. The neural network model is combined by [32] with a domain dictionary to improve cyber security entity recognition by constructing the domain dictionary for correction. Supervised learning is used by [33] to derive entity recognition methods for cyber security essay. Furthermore, it is worth emphasizing that the field of requirements engineering is increasingly recognizing the significance of zero-shot and few-shot learning approaches. Alhosan et. al. [34], proposed a zero-shot requirements classifier that addresses the challenge of limited labeled training data in requirements engineering. With respect to the NER, Malik et.al. [5] explored a few-shot NER approach and supervised approaches to extract software-specific entities from SRS documents.

The comparison of state-of-the-art NER methods with our approaches in software engineering is depicted in Table XIII.

VII. Conclusion

When compared to the state-of-the-art approaches, both of our methods outperformed existing supervised named entity recognition models in this specific domain.

In future, it would be interesting to analyze the trade-offs between zero-shot and supervised learning approaches. Further improvement of accuracy and efficiency of these zero-shot models may be achieved by incorporating additional information sources as domain knowledge. It would also be interesting to examine the generalizability of these approaches to different languages and contexts.

VIII. Acknowledgments

Work partially supported by iNEST Interconnected NordEst Innovation Ecosystem, funded by PNRR (Mission 4.2, Investment 1.5), NextGeneration EU—Project ID: ECS 00000043, and by SPIN2021 Ressa_Rob, funded by Ca'Foscari University.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] A. Ferrari, P. Spoletini, and S. Gnesi, "Ambiguity and tacit knowledge in requirements elicitation interviews," *Requirements Engineering*, vol. 21, no. 3, pp. 333–355, 2016.

[2] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated extraction and clustering of requirements glossary terms," *IEEE Transactions on Software Engineering*, vol. 43, no. 10, pp. 918–945, 2016.

[3] G. B. Herwanto, G. Quirchmayr, and A. M. Tjoa, "A named entity recognition based approach for privacy requirements engineering," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2021, pp. 406–411.

[4] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[5] G. Malik, M. Cevik, S. Bera, S. Yildirim, D. Parikh, and A. Basar, "Software requirement-specific entity extraction using transformer models," in *The 35th Canadian Conference on Artificial Intelligence*, 2022.

[6] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, "Software-specific named entity recognition in software engineering social content," in *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 90–101.

[7] T. Gemkow, M. Conzelmann, K. Hartig, and A. Vogelsang, "Automatic glossary term extraction from large-scale requirements specifications," in *2018 IEEE 26th International Requirements Engineering Conference (RE)*. IEEE, 2018, pp. 412–417.

[8] J. Tabassum, M. Maddela, W. Xu, and A. Ritter, "Code and named entity recognition in stackoverflow," *arXiv preprint arXiv:2005.01634*, 2020.

[9] C. Zhou, B. Li, and X. Sun, "Improving software bug-specific named entity recognition with deep neural network," *Journal of Systems and Software*, vol. 165, p. 110572, 2020.

[10] N. Ding, S. Hu, W. Zhao, Y. Chen, Z. Liu, H. Zheng, and M. Sun, "Openprompt: An open-source framework for prompt-learning," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, ACL 2022*. Association for Computational Linguistics, 2022, pp. 105–113.

[11] C. Gao, X. Zhang, and H. Liu, "Data and knowledge-driven named entity recognition for cyber security," *Cybersecurity*, vol. 4, no. 1, pp. 1–13, 2021.

[12] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[13] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–37, 2019.

[14] B. Romera-Paredes and P. Torr, "An embarrassingly simple approach to zero-shot learning," in *International conference on machine learning*. PMLR, 2015, pp. 2152–2161.

[15] A. Parnami and M. Lee, "Learning from few examples: A summary of approaches to few-shot learning," *arXiv preprint arXiv:2203.04291*, 2022.

[16] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, Volume 1*, 2018, pp. 1112–1122.

[17] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," *arXiv preprint arXiv:1705.02364*, 2017.

[18] J. Bos and K. Markert, "Recognising textual entailment with logical inference," in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 2005, pp. 628–635.

[19] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[20] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer." *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[21] J. P. McCrae, A. Rademaker, F. Bond, E. Rudnicka, and C. Fellbaum, "English wordnet 2019–an open-source wordnet for english," in *Proceedings of the 10th Global WordNet Conference*, 2019, pp. 245–252.

[22] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL-HLT 2019, Volume 1*. ACL, 2019, pp. 4171–4186.

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

[24] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL, 2020, pp. 7871–7880.

[25] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Volume 2*. ACL, 2018, pp. 784–789.

[26] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016*. The Association for Computational Linguistics, 2016, pp. 2383–2392.

[27] A. Dogan and D. Birant, "A weighted majority voting ensemble approach for classification," in *2019 4th International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2019, pp. 1–6.

[28] A. Wang and K. Cho, "Bert has a mouth, and it must speak: Bert as a markov random field language model," *arXiv preprint arXiv:1902.04094*, 2019.

[29] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *arXiv preprint arXiv:2203.02155*, 2022.

[30] M. Veera Prathap Reddy, P. Prasad, M. Chikkamath, and S. Mandadi, "Nerse: named entity recognition in software engineering as a service," in *Service Research and Innovation*. Springer, 2018, pp. 65–80.

[31] M. Tikhomirov, N. Loukachevitch, A. Sirotina, and B. Dobrov, "Using bert and augmentation in named entity recognition for cybersecurity domain," in *International Conference on Applications of Natural Language to Information Systems*. Springer, 2020, pp. 16–24.

[32] H. Wu, X. Li, and Y. Gao, "An effective approach of named entity recognition for cyber threat intelligence," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1. IEEE, 2020, pp. 1370–1374.

[33] N. Dionísio, F. Alves, P. M. Ferreira, and A. Bessani, "Cyberthreat detection from twitter using deep neural networks," in *2019 international joint conference on neural networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[34] W. Alhoshan, A. Ferrari, and L. Zhao, "Zero-shot learning for requirements classification: An exploratory study," *Information and Software Technology*, vol. 159, p. 107202, 2023.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 18:03:18 UTC from IEEE Xplore. Restrictions apply.