Canadian AI 2022
30 May–3 June 2022, Toronto

Software requirement-specific entity extraction using transformer models

Garima Malik[†], Mucahit Cevik[†, *], Swayami Bera[†], Savas Yildirim[†, ◇], Devang Parikh[‡], Ayse Basar[†]

[†] Ryerson University, Toronto, Canada
[‡] IBM, North Carolina, USA
[◇] Istanbul Bilgi University, Istanbul, Turkey

Abstract

Software Requirement Specifications (SRS) documents provide the description of requirements and expectations attributed to software products. The structured text present in the SRS documents serves as a guide for developers in defining various functions in the process of software development. Software specific entity extraction is an important pre-processing step for various Natural Language Processing (NLP) tasks in the requirement engineering domain such as entity-centric search systems, SRS document summmarization, requirement classification, and requirement quality management. Recent advances in transformer-based models have significantly contributed to NLP and information retrieval problems, and achieved state-of-the-art performance for domain specific entity extraction tasks. In this study, we employ the transformer models including BERT, RoBERTa and ALBERT for software specific entity extraction. For this purpose, we annotate three requirement datasets, namely, DOORS, SRE, and RQA with varied sets of software specific entities. Our numerical study shows that transformer models are able to outperform the traditional approaches such as ML-CRF, and we find that BERT variants improve the F1-scores by 4% and 5% on the DOORS and SRE datasets, respectively. We conduct entity level error analysis to examine the partial and exact matching of entities and respective boundaries. Lastly, we experiment with few-shot learning to create sample efficient NER systems with template-based BART model.

Keywords: Software-specific entity extraction, Transformer models, Few-shot learning, Software Requirement Specifications (SRS)

1. Introduction

Software Requirements Specifications (SRS) documents comprise a road map of the project that is being developed. It details the scope, the budget constraints, intended use-cases, the functionalities to be expected, the sequence of actions to be performed and the obstacles to overcome. SRS documentation is an important aspect of any product development to ensure that everyone is in accord with the plan of action, from the stakeholders to the development team.

The content of the SRS documents is curated in such a way that it is comprehensible for all the personnel involved in the software development process, and humans in general. Machines, on the other hand, generally cannot make sense of these documents that are text-heavy and filled with technical jargon. Accordingly, for automation purposes, these are a collection of structured natural language texts and, in order to discern the contents of these documents, we can employ information extraction or entity identification systems.

Named entity recognition (NER) is a type of information retrieval technique that aims to find and sort tokens in the text into predefined categories. Applied to the SRS documents, the named entities would be software specific designations such as actor, action, operator, user, object, GUI, hardware, API, and metric. Executing NER on SRS documents works as a precursor to various entity-centric applications such as requirement classification, software document classification, text summarization, data analysis, use case generation, and topic modeling.

[*] mcevik@ryerson.ca

Research motivation. SRS documents can be perceived as well-structured text which explain the functional and non-functional requirements of a system. Extraction of software specific entities can provide us with a high-level overview of SRS documents and this information can be further analysed to group similar and dissimilar requirements. Specifically, these software specific entities can be utilised as an additional feature in requirement quality assessment systems to improve the quality of software requirements. Most importantly, we can train machine learning models with requirement texts and software-specific entities as features for various natural language processing (NLP) tasks in requirement engineering domain. In this study, we explore the capabilities of transformer-based models for software specific entity extraction.

Contribution. The main contributions of our study can be summarized as follows:

- We implement machine learning-based ML-CRF [1] and widely used transformer models including BERT [2], RoBERTa [3], and ALBERT [4] over three requirement datasets for software specific entity extraction. To the best of our knowledge, previous studies did not consider transformer models in software-specific entity extraction.
- We conduct a detailed numerical study by considering five different NER models and three distinct annotated requirement datasets with varying software-specific entity sets. These datasets are obtained from open-source SRS documents associated with different industries, such as aerospace, automobile, healthcare, and transportation. We also present entity level error analysis to capture the partial entity and boundary matches in a software-specific entity extraction task.
- We investigate few-shot learning methods for the NER task. We implement the text-to-text language model BART [5], which restructures the requirements into input text and candidate text, and predicts the software specific entity tags for the given input. We find that few-shot learning performs better than the BERT variants for smaller training sets, however, this comes at the expense of increased training time.

2. Literature review

Only a few studies consider NLP models for NER tasks in the field of software engineering. The SoftNER model presented by Tabassum et al. [6] is a BERT-based model modulated to work well on StackOverflow data to identify code tokens and software-related token labels. RucyBERT is a BERT model trained to tackle textual data specific to the cyber-security domain [7]. DBNER use a deep neural network model for bug-specific entity recognition [8]. It employs a combination of bidirectional LSTM (BiLSTM) and conditional random field (CRF) models to learn many features from large amounts of data extracted from bug repositories, and incorporates an attention layer to refine the entity recognition process.

In a recent study, Nayak, Kesri, and Dubey [9] presented a knowledge graph-based tool that generates requirement text instances from software engineering documents. They proposed a constituency parse tree-based path finding algorithm for test intent extraction, and employed a CRF-based model with automatic feature engineering for the NER task. Ye et al. [10] developed a software-specific NER method (S-NER) and identified the design challenges in creating a NER methodology for social content data. They annotated Stack Overflow posts with five tags namely, API, platform, software standard, programming language, and tools. They trained an ML-CRF model with software-specific gazetteers and unsupervised word clusters. Reddy et al. [11] defined a set of 22 software-specific entities in Stack Overflow posts which includes programming languages (e.g., web development and

scripting languages), names of software tools, frameworks and protocols. Their numerical study revealed that BiLSTM-CRF performed better than ML-CRF for their NER task.

Many previous studies have used few-shot learning for NER when the number of in-domain labeled data instances is very low. Yang and Katiyar [12] demonstrated impressive results for standard few-shot learning applications with a model using nearest neighbor learning and structured inference. Das et al. [13] proposed CONTaiNER, a new contrastive learning method that enhances the inter-token distribution distance for using the few-shot setting on NER. FewNER made use of a meta-learning technique, and it applied novel N-way K-shot learning approach for applying few-shot learning to NER [14].Fritzler, Logacheva, and Kretov [15] used a metric learning approach called Prototypical Networking which learnt the intermediate representations of tokens that fall under the same named entity category. This method allows the classification of tokens with very few training instances and also shows promise to be used as a zero-shot learning method. Text-to-text language models such as BART and T5 have the ability to transform original sentences into proper useful statements for the NER tasks, as can be seen in the examples "Paris is a type of *city*", "John is a type of *person*". Cui et al. [5] utilized text-to-text BART model for NER in few-shot mode. They treated NER as text-to-text and a language model ranking problem. That is, input sentences and candidate statement templates are ranked and used to produce named entity candidates based on the ranking scores.

3. Methodology

In this section, we first describe the three requirement datasets annotated with software-specific entities. Then, we briefly summarize the machine learning and transformer-based models used for NER, and we explain the experimental setup for our numerical study. Figure 1 shows the process of training the NER models with software requirements as input.

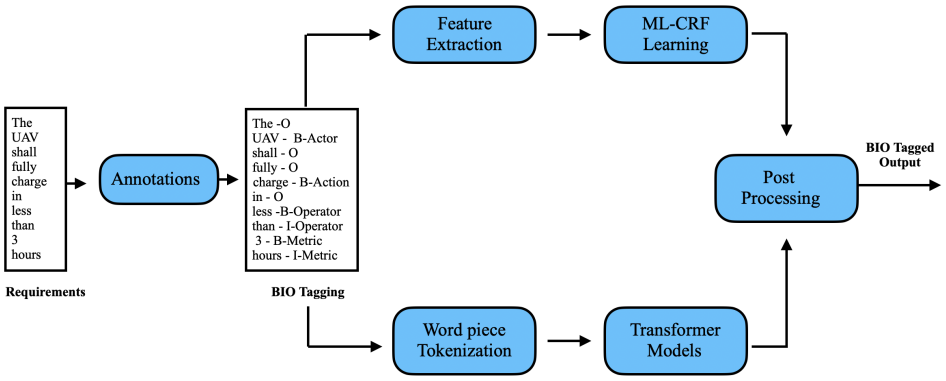


Figure 1. An overview of the work flow for transformers- and ML-CRF-based NER models

3.1. Data preparation

We consider three software engineering-specific datasets comprised of software requirements extracted from different SRS documents. We define different sets of software-specific entities in these datasets, and implement a BIO tagging scheme where "B" indicates the

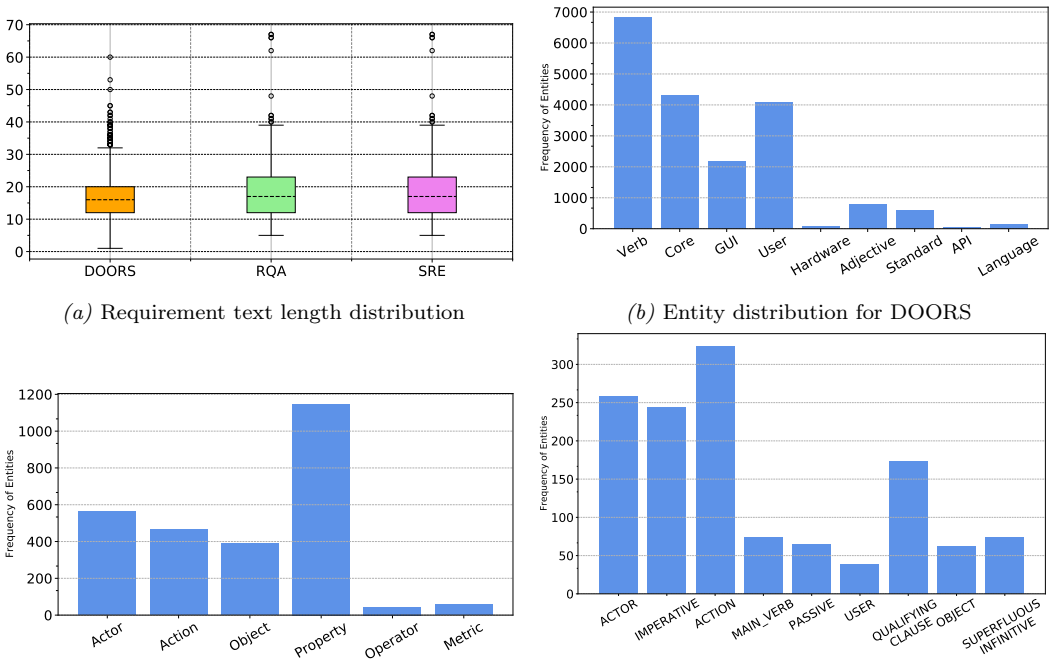
first word of an entity, "I" indicates words inside of an entity , and "O" indicates non-entity words (see Figure 1). Below, we briefly describe the datasets used in our analysis.

- ***DOORS***: Dynamic Object Oriented Requirements System (DOORS) is a requirement management tool [16]. It aids the process of requirement collection, and managing, verifying and communicating requirements through the process of software development in real time. Initially, we extracted 5,100 DOORS user stories (shorter version of the requirements) related to multiple software projects. These requirements were subjected to data cleaning and basic pre-processing, which resulted in 3,333 requirements. We employed both human experts with software development background and rule-based system in IBM Watson Studio[1] software to obtain the complete annotations. DOORS requirements consist of ten software specific entities where 'verb' is the most dominant and 'API' is the least frequent entity (see Table 1 and Figure 2b).
- ***SRE***: Software Requirement Entities (SRE) is an extensive dataset that is prepared from five different SRS documents attributed to different industries such as UAV (Aerospace) [17], OpenCoss (Transportation)[2], WorldVista (Medical)[3], Mashbot (Robotics) [18] and Thermostat (Thermodynamics) [18]. Each one of these SRS documents contain more than 100 requirements. To create the SRE dataset, we sampled 378 requirements and converted the compound requirement structures into simple requirements. The annotation process was accomplished by human experts using Doccano [4] and IBM Watson Studio[1]. This dataset consists of six software specific entities with 'Property' being the most frequent and 'Operator' being the least frequent (see Figure 2c). Due to the dominance of hardware related requirements, we encountered some mathematical measures ('3 hours' or '70 Km') and operators ('less than' or 'equals to') in the text. Accordingly, we kept the 'Metric' and 'Operator' as entities to represent this information. Figure 2a shows that the longest requirement text length is 67 and shortest is 7.
- ***RQA***: Requirement Quality Assistant (RQA) is one of the inherent functionality provided by DOORS which improves the quality of software requirements [19]. It uses the International Council on Systems Engineering (INCOSE) guidelines for writing the requirements and helps in formulating the good quality requirements [20]. With the help of the RQA system, users can also assess the quality of requirements using NLP-based feature extraction. The RQA system detects 20 software specific entities which cover both linguistic and morphological aspects of requirements. We obtained the corresponding annotations for the requirements present in SRE dataset and created another software-specific NER dataset, which we refer to as RQA dataset. Figure 2d shows the nine most frequent entities present in this dataset.

Table 1. Software requirement dataset characteristics

Dataset	# of Requirements	# of Entities
DOORS	3,333	10
SRE	378	6
RQA	378	20

[1]https://www.ibm.com/ca-en/cloud/watson-studio
[2]http://www.opencoss-project.eu
[3]http://coest.org/datasets
[4]https://github.com/doccano/doccano



(a) Requirement text length distribution

(b) Entity distribution for DOORS

(c) Entity distribution for SRE

(d) Entity distribution for RQA

Figure 2. Exploratory data analysis on software requirement datasets

3.2. **Named entity recognition models**

BERT models are shown to achieve state-of-the-art performance for the domain specific NER tasks, and it is very convenient to fine-tune these models for downstream tasks. As such, we employ BERT variants, namely, BERT (cased and uncased), RoBERTa and ALBERT, along with the baseline ML-CRF model for our NER task.

- ***ML-CRF***: Machine Learning-based Conditional Random Fields (ML-CRF) belongs to the class of statistical models that use a probabilistic approach to assign the entity tags for each token present in the input text. Previous studies showed that ML-CRF can be highly effective for domain specific NER tasks as it can take advantage of data-specific handcrafted features that capture the contextual information in the input text [9–11, 21]. For our analysis, we prepared a comprehensive list of handcrafted features which includes orthographic (capitalization and alphanumeric), lexical (tokens), contextual (tokens in window size [-1,1]), and word bitstring (digits and alphabets). This feature set is inputted to the CRF learning model which estimates the conditional probability of selecting appropriate software specific entities given the input as a requirement text (see Figure 1).
- ***BERT:*** BERT is a language representation model which stands for Bidirectional Encoder Representations from Transformers [2]. It is pre-trained on a large corpus of data using masked language modeling. We used both BERT-large-cased and BERT-large-uncased versions in our analysis. The main difference in the uncased version is that the text has been transformed into lower case before the process of tokenization, whereas the cased version performs the training process with raw text.
- ***RoBERTa:*** RoBERTa is a robustly optimized version of BERT. It uses the BERT-large variant as an underlying structure. The modifications over vanilla BERT includes training the model longer with large batches, removing the next sentence

prediction objective, training on larger sequences, and dynamically changing the masking pattern in modeling [3].

- ***ALBERT:*** ALBERT is the *lite* variant of BERT [4]. It simplifies the BERT architecture by reducing the number of parameters to optimize the memory efficiency. Specifically, the ALBERT model changed the token embedding layer units from 768 to 128 which lead to an increase in training speed. In our analysis, we used ALBERT-base-V2 checkpoint.

3.3. **Few-shot learning**

Few-shot learning refers to making classification based on a small number of data instances. One of the leading methods for few-shot is prompt-based learning, which has become a new paradigm in the NLP field due to its simplicity. This approach involves reformulating downstream tasks as Masked Language Model (MLM) of a pre-trained language model, showing great effectiveness for various NLP tasks. For instance, a NER task can be formulated as "John went to Paris. Paris is a kind of [MASK].". When this instance is passed to MLM, the output is likely to produce city, place or town. Accordingly, we converted the software requirement texts into input sentences and candidate statements. For instance "*UAV shall charge in 3 hours*" can be represented in the form of candidate labels as '*UAV is a type of actor*', '*charge is a type of action*', and '*3 hours is a type of metric*'. We trained the template-based BART model [5] with transformed version of the software requirement datasets. The input of the text-to-text (BART) model can be perceived as software requirements, and output is the candidate statements which represents the named entity labels.

3.4. **Experimental setup**

In a named entity extraction task, the main aim is to identify the correct tag for each token present in the input text. We employed different BERT variants for this task and instantiated pre-trained transformer models (i.e., BERT-large-uncased, BERT-large-cased, and RoBERTa-large). We fine-tuned the architectures for the NER task by using the *AutoModelForTokenClassification* class in the "simpletransformers.classification" library, which is based on the transformers library by HuggingFace [5].

The "Simple Transformers" library provides a convenient way of training and evaluating the transformer models. The process of fine-tuning the transformer models is typically time consuming. However, this can be mitigated by altering the length of the input text, which is the maximum length of tokens that appear in the model corpus. We used 100 as the default value for the max_seq_length parameter since the requirement length ranges from (7 to 70) in all the datasets. Because the number of annotated requirements is limited in all three datasets, we employed 3-fold cross validation to evaluate the performance of the NER models.

Table 2 shows the hyperparameter values used in training the NER models. We trained the ML-CRF model using CRFsuite library [6] and performed hyperparameter tuning using the grid search cross validation technique. In Table 2, 'C1' and 'C2' correspond to L1 and L2 regularization, respectively. We used 'lbfgs', i.e., limited memory-BFGS as the optimization algorithm, and trained the algorithm for 100 iterations.

Similarly, we experimented with different hyperparameters for the transformer models to identify the ideal parameter set. Table 2 shows the values of fine-tuned parameters for each BERT variant. For hyperparameter tuning experiments, we set up different combinations

[5]https://huggingface.co/transformers/model_doc/bert.html
[6]https://sklearn-crfsuite.readthedocs.io/en/latest/

of learning rate (1e-05 to 5e-05), training epochs (2,5), train batches (8,32) and dropout value (0.1). We found that transformer models achieve better performance with smaller training batches and converge within a few epochs. We measured the average training time for all the requirement datasets, and found ML-CRF to be the fastest NER model with an average training time of 21 seconds. Average training times for the transformer models are as follows: 9.12 minutes for BERT-large-cased, 9.43 minutes for RoBERTa-large, and 2.9 minutes for ALBERT-base-v2.

Table 2. Model configurations and hyperparameters for all the models

Models	Hyperparameters
ML-CRF	algorithm : 'lbfgs',c1 : 0.1, c2 : 0.1, max_iterations : 100
BERT-large-uncased	num_hidden_layers : 16 hidden_size : 1024, max_seq_length : 100, epoch : 5, drop_out : 0.1, learning_rate : 5e-05,batch_size : 8
BERT-large-cased	num_hidden_layers : 16, hidden_size :1024, max_seq_length : 100, epoch : 5, drop_out : 0.1, learning_rate : 5e-5,batch_size : 8
RoBERTa-large	num_hidden_layers : 16, hidden_size : 1024, max_seq_length : 100, epoch : 3, drop_out : 0.1, learning_rate : 5e-04,batch_size : 8
ALBERT-base-v2	Repeating_layers : 12, hidden_size : 768, max_seq_length : 100, epoch : 5, drop_out : 0.1, learning_rate : 2e-5, batch_size : 8

4. **Results**

In this section, we first compare five NER models over three software requirement datasets (DOORS, RQA and SRE) using the standard NER performance metrics, namely, weighted versions of precision, recall, F1-score, and accuracy. The weighted-averaged F1-score is measured by taking the mean of all individual entity F1-scores weighted by their support, where support refers to the number of true instances for each entity. We next present an entity level error analysis to investigate underlying causes and remedies to improve NER performance. Lastly, we provide few-shot learning results using DOORS, which is our largest dataset.

4.1. **Entity extraction performance**

Table 3 shows the comparison of transformer models with the baseline ML-CRF model. The reported values represent the metrics in percentages accompanied with standard deviation over 3 folds of data. Previous studies show that transformer models achieve state-of-the-art performance for various domain specific NER tasks, given sufficient number of instances and fine-tuning of the model parameters [6, 22]. We observe a similar trend in our results as well. We find that the BERT-large-cased version consistently outperforms the ML-CRF model for DOORS and SRE dataset with F1-score of 92.6% and 77.3% respectively. That is, we manage to improve the F1-score and accuracy by 4% and 5% for DOORS and SRE dataset with the BERT-large-cased model. We note that, in software requirement texts, capitalization and abbreviations play an important role, which helps explaining the better performance of the 'cased' BERT version compared to other transformer models.

ML-CRF achieves the best performance for the RQA dataset, with an F1-score of 73.4%. With fewer input instances and larger entity sets available in this dataset, BERT models are not able to achieve the expected performance. For some of the entities in the RQA dataset, we observe minor support values in the training set which make it difficult for the BERT models to identify the required patterns for the entities.

Table 3. Entity extraction performance values for the NER models.

	DOORS			
Models	Acc(%)	P(%)	R(%)	F1(%)
ML-CRF	92.9 ± 0.00	88.0 ± 0.00	89.6 ± 0.00	88.7 ± 0.00
BERT-large-cased	**95.8 ± 0.00**	**91.3 ± 0.01**	**94.3 ± 0.00**	**92.6 ± 0.00**
BERT-large-uncased	95.2 ± 0.00	89.3 ± 0.00	94.0 ± 0.00	91.3 ± 0.00
RoBERTa-large	95.3 ± 0.00	90.0 ± 0.01	94.0 ± 0.00	92.0 ± 0.00
ALBERT-base-v2	94.9 ± 0.00	89.6 ± 0.02	92.6 ± 0.00	91.0 ± 0.01
	SRE			
Models	Acc(%)	P(%)	R(%)	F1(%)
ML-CRF	84.2 ± 0.00	75.4 ± 0.03	69.8 ± 0.00	72.3 ± 0.01
BERT-large-cased	**89.1 ± 0.04**	**74.3 ± 0.07**	**81.3 ± 0.07**	**77.3 ± 0.07**
BERT-large-uncased	88.4 ± 0.03	71.0 ± 0.07	81.3 ± 0.05	75.6 ± 0.06
RoBERTa-large	86.5 ± 0.02	68.6 ± 0.04	77.0 ± 0.04	72.3 ± 0.05
ALBERT-base-v2	88.1 ± 0.03	74.0 ± 0.06	77.0 ± 0.06	75.3 ± 0.06
	RQA			
Models	Acc(%)	P(%)	R(%)	F1(%)
ML-CRF	**91.0 ± 0.00**	**81.5 ± 0.02**	**69.0 ± 0.00**	**73.4 ± 0.00**
BERT-large-cased	90.6 ± 0.00	62.0 ± 0.01	67.6 ± 0.03	64.6 ± 0.04
BERT-large-uncased	90.2 ± 0.00	61.6 ± 0.06	65.0 ± 0.01	61.6 ± 0.03
RoBERTa-large	89.6 ± 0.00	53.3 ± 0.03	63.3 ± 0.03	58.0 ± 0.02
ALBERT-base-v2	91.0 ± 0.01	73.3 ± 0.02	70.0 ± 0.04	69.3 ± 0.04

Table 4 shows the entity specific evaluation with the best performing model for each dataset. BERT-large-cased model performs well for DOORS over all the entities except 'Language' entity, as we have very few instances for this category in the training set. It achieves very high F1-scores for entities such as 'User'(97.6%), 'Verb'(95%), and 'API' (95.3%) along with low standard deviation values.

Table 4. Detailed performance values for individual entities with the best performing model

Dataset	Entities	P(%)	R(%)	F1(%)	Support
DOORS	API	93.3 ± 0.01	97.0 ± 0.00	95.3 ± 0.01	33
	Adjective	87.3 ± 0.02	91.3 ± 0.02	89.6 ± 0.02	281
	Core	88.6 ± 0.02	93.0 ± 0.00	91.0 ± 0.01	1,237
	GUI	83.0 ± 0.02	88.3 ± 0.01	85.6 ± 0.01	592
	Hardware	86.3 ± 0.04	88.3 ± 0.10	87.0 ± 0.07	26
	Language	72.6 ± 0.03	59.6 ± 0.15	64.3 ± 0.09	37
	Platform	89.3 ± 0.00	92.0 ± 0.01	91.0 ± 0.00	235
	Standard	89.3 ± 0.01	93.0 ± 0.01	91.0 ± 0.01	235
	User	97.0 ± 0.01	98.0 ± 0.00	97.6 ± 0.00	923
	Verb	93.0 ± 0.01	96.6 ± 0.00	95.0 ± 0.00	2607
SRE	Action	85.0 ± 0.06	86.6 ± 0.08	86.0 ± 0.06	150
	Actor	78.3 ± 0.04	92.6 ± 0.05	85.0 ± 0.04	146
	Metric	92.6 ± 0.06	91.6 ± 0.09	92.0 ± 0.07	12
	Object	65.6 ± 0.10	64.6 ± 0.09	65.3 ± 0.10	59
	Operator	54.6 ± 0.33	29.3 ± 0.20	38.0 ± 0.25	11
	Property	64.0 ± 0.11	76.0 ± 0.08	69.6 ± 0.10	218
RQA	Action	82.0 ± 0.04	74.0 ± 0.04	78.0 ± 0.03	100
	Actor	85.0 ± 0.00	79.0 ± 0.05	81.0 ± 0.02	81
	Ambiguity	71.0 ± 0.15	53.0 ± 0.25	59.0 ± 0.19	10
	Clause	95.0 ± 0.03	89.0 ± 0.09	92.0 ± 0.04	24
	Combinators	61.0 ± 0.28	15.0 ± 0.01	23.0 ± 0.01	6
	Design_statements	100.0 ± 0.00	86.0 ± 0.18	91.0 ± 0.11	3
	Escape_clause	100.0 ± 0.00	100.0 ± 0.00	100.0 ± 0.00	1
	Imperative	87.0 ± 0.03	85.0 ± 0.02	86.0 ± 0.01	81
	Qualifying_clause	79.0 ± 0.07	60.0 ± 0.02	68.0 ± 0.01	54
	Superfluous_infinitive	100.0 ± 0.00	78.0 ± 0.10	87.0 ± 0.06	10
	User	71.0 ± 0.01	63.0 ± 0.05	67.0 ± 0.03	12

With limited amount of available data for SRE and RQA datasets, NER models are not able to perform well for certain entities. For the SRE data, the BERT-large-cased is able to perform well for entities 'Action', 'Actor', and 'Metric' with the F1-scores of 86%, 85%, and 92%, respectively. BERT-large-cased provides average performance values with high standard deviation for 'Operator' and 'Property' entity. The low performance values for these entities can be explained by lack of uniformity in requirement structures as the 'Property' tag was defined differently in all the SRS documents because of the variety of writing styles of requirements, and 'Operator' tag support was small. As the entity set is very large for the RQA dataset, we get zero support for some of the entities during evaluation. Therefore, we only present the best 11 entities out of 20 entities in Table 4. ML-CRF works better for most entities, irrespective of low support, as it trains over a customised feature set for each entity.

4.2. **Error analysis**

A NER task is typically formulated in two steps: first identifying the boundary of tags (i.e., recognizing the beginning and end of the tag), and then assigning the appropriate entity. As such, we explore possible cases of errors in NER task both with respect to boundary matching and entity type matching, and report the error measures on our datasets. Table 5 describes five scenarios of errors, which can happen in model predictions with respect to the ground truth annotations for sample requirements from the SRE dataset. Case I and III can be easily evaluated by classic NER metrics. However, for partial entity and boundary matches, we need different sets of metrics to evaluate the scenarios described in cases II, IV and V.

Table 5. Sample error cases from the SRE dataset with the predictions from the BERT-large-cased model [23].

Cases	Description	Ground truth	Prediction
I	Surface string and entity type match	New user will require a username to create the account. O, B-Actor, O, B-Action, O, B-Property O,B-Action, O, B-Property	New user will require a username to create the account. O, B-Actor, O, B-Action, O, B-Property O,B-Action, O, B-Property
II	System hypothesized an entity	UAV shall fully charge in 3 hours. [B-Actor, O, O, B-Action, O B-Metric, I-Metric]	UAV shall fully charge in 3 hours. [B-LOC, O,O, B-Action, O, B-Metric, I-Metric]
III	System misses an entity	Driver shall control the tame gap between vehicles. [B-Actor, O, B-Action, O, B-Property I-Property, O, B-Object]	Driver shall control the tame gap between vehicles [O,O,O, B-Property, I-Property O, B-Object]
IV	System get the boundary wrong	The Themas System shall note the temperature. [O, B-Actor, I-Actor, O, B-Action, O, B-Property]	The Themas System shall note the temperature. [B-Actor, I-Actor, I-Actor, O, B-Action ,O, B-Property]
V	System get the boundary and entity type wrong	The system shall report the temperature of heating and Cooling units. [O, B-Actor, O, B-Action, O, B-Property, O B-Object, I-Object, I-Object, I-Object]	The system shall report the temperature of heating and Cooling units. [O, B-Actor, O, B-Action, O, B-Property, O B-Property, O, I-Property,O]

Message Understanding Conference (MUC) introduced the set of error metrics in an assessment that can be explained in terms of comparing the predictions of a NER model against the ground truth[24]:

- Correct (C): entity type and boundary are the same (Case I)
- Incorrect (I): output of a system and the golden standard are not same (Case V)
- Partial (P): system and the golden standard are partially "similar" but not the same (Case IV)
- Missing (M): entity type is not captured by system (Case III)
- Spurius (S): system produces an entity which is not present in the entity set (Case II)

The Semantic evaluation workshop'13 (SemEval'13) conceptualised four different ways to measure precision and recall based on the metrics defined by MUC [23]:

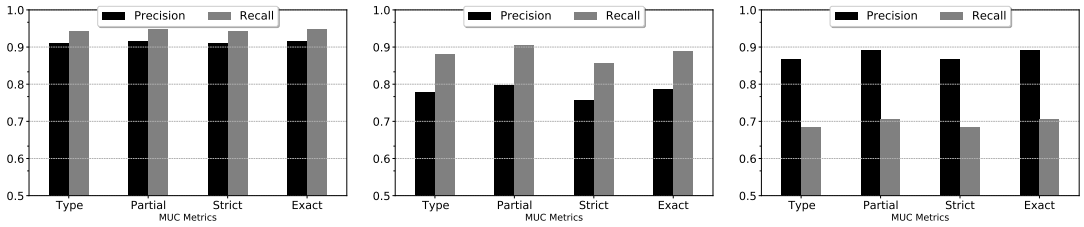
- Strict: exact match for boundary and entity type
- Exact: exact boundary match irrespective of entity type match
- Partial: partial boundary match irrespective of entity type match
- Type: entity type match irrespective of the boundary match

The total number of entities present in the ground truth can be defined as 'Possible (POS)' ($POS = C + I + P + M$) measure, and total number of system predictions can be defined as 'Actual (ACT)' ($ACT = C + I + P + S$). For exact and strict type of errors listed in SemEval[23] we define precision and recall values as $Precision = \frac{C}{ACT}$ and $Recall = \frac{C}{POS}$, and for partial or type match errors as $Precision = \frac{C+0.5\times P}{ACT}$ and $Recall = \frac{C+0.5\times P}{POS}$. To calculate these measures, we consider the best test folds of DOORS, SRE, and RQA having 1253, 126, 126 instances. We extract the system predictions using the best NER model for each dataset and compare them with ground truth annotations. Table 6 presents the error values for each dataset in absolute number. The NER models miss 5.0%, 8.1% and 29.4% entities in DOORS, SRE and RQA, respectively. In terms of spurious error, the NER models assign 8.2%, 19.0% and 10.8% of random entities to tokens present in DOORS, SRE, and RQA, respectively.

Table 6. ACT (number of entities produced by NER system) and POS (expected number of entities) values obtained using five different types of error measures as defined by [24].

Dataset		#Correct (C)	#Incorrect (I)	#Partial (P)	#Missed (M)	#Spurious (S)	#Possible (POS)	#Actual (ACT)
DOORS	Type	5968.0 (94.0%)	40.0 (0.6%)	0.0 (0.0%)	320.0 (5.0%)	541.0 (8.2%)	6328.0	6549.0
	Partial	6002.0 (94.8%)	0.0 (0.0%)	6.0(0.09%)	320.0 (5.0%)	541.0 (8.2%)	6328.0	6549.0
	Strict	5964.0 (94.2%)	44.0 (0.7%)	0.0 (0.0%)	320.0 (5.0%)	541.0 (8.2%)	6328.0	6549.0
	Exact	6002.0 (94.8%)	6.0 (0.09%)	0.0 (0.0%)	320.0 (5.0%)	541.0 (8.2%)	6328.0	6549.0
SRE	Type	531.0 (88.2%)	22.0 (3.6%)	0.0 (0.0%)	49.0 (8.1%)	129.0 (19.0%)	602.0	682.0
	Partial	536.0 (89.0%)	0.0 (0.0%)	17.0 (2.8%)	49.0 (8.1%)	129.0 (19.0%)	602.0	682.0
	Strict	516.0 (85.7%)	37.0 (6.1%)	0.0 (0.0%)	49.0 (8.1%)	129.0 (19.0%)	602.0	682.0
	Exact	536.0 (89.0%)	17.0 (2.8%)	0.0 (0.0%)	49.0 (8.1%)	129.0 (19.0%)	602.0	682.0
RQA	Type	312.0 (68.5%)	9.0 (1.9%)	0.0 (0.0%)	134.0 (29.4%)	39.0 (10.8%)	455.0	360.0
	Partial	321.0 (70.5%)	0.0 (0.0%)	0.0 (0.0%)	134.0 (29.4%)	39.0 (10.8%)	455.0	360.0
	Strict	312.0 (68.5%)	9.0 (1.9%)	0.0 (0.0%)	134.0 (29.4%)	39.0 (10.8%)	455.0	360.0
	Exact	321.0 (70.5%)	0.0 (0.0%)	0.0 (0.0%)	134.0 (29.4%)	39.0 (10.8%)	455.0	360.0

Figure 3 shows the precision and recall values for each error category for all the datasets. For DOORS data, we observe similar precision (91%) and recall (94%) values, however, for SRE dataset we get higher values of recall across all the error categories ranging from (88% to 90%), with precision values ranging from (77% to 79%). Because the RQA dataset contains a large entity set and limited number of data instances, we obtain lower recall values (68% to 70%) in all the error categories.



(a) DOORS *(b)* SRE *(c)* RQA

Figure 3. Precision and recall values obtained after evaluating the error categories [23].

4.3. **Few-shot learning results**

It is usually difficult to obtain labeled textual data in software engineering domain. Accordingly, it is important to test out the capabilities of the NLP models with limited amount

of available data. We used the BART-based approach [5] as the backbone in few-shot training, and compare its performance against vanilla BERT fine-tuning. Respectively, 4% ,8%, 16%, 32%, 64% and 100% of the entire DOORS dataset used in a log-scale mode. Figure 4 demonstrates that the few-shot text-to-text model can provide relatively good performance with a small number of labeled data instances. That is, few-shot learning is substantially more sample efficient than vanilla BERT Fine-tuning. Even at only 16% of the original training set, the model achieves 82.5% accuracy value, which outperforms BERT (78.7%) significantly. On the other hand, BERT outperforms few-shot learning when all the available data is used for training. Furthermore, the time complexity of the Template-NER model used for few-shot learning is very high. The model enumerates all possible text spans in the original input sentence as named entity candidates during inference. Afterward, it classifies the candidates into named entities or non-entities based on model scores. Enumerating all possible text inevitably increases time complexity.

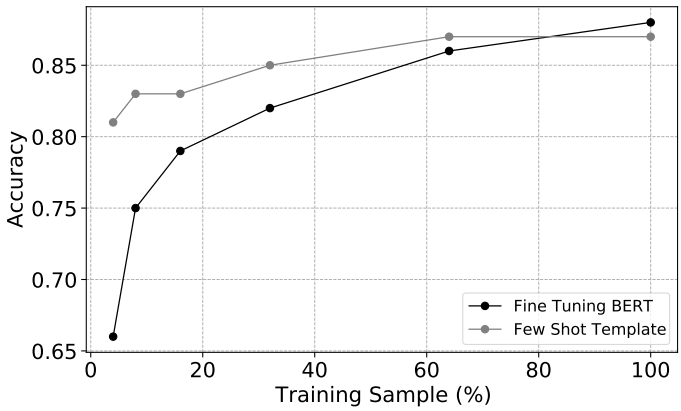


Figure 4. Few-shot and vanilla BERT accuracy as a function of training set size

5. **Conclusions**

In this study, we demonstrated the efficiency of transformers models for the extraction of software-specific entities using three requirement datasets (DOORS, SRE, and RQA). We employed four different BERT variants, and compared them with the baseline ML-CRF model. In addition to standard NER metrics, we provided entity level error analysis to examine the error cases in NER model predictions.

In general, BERT consistently provided the best performance for DOORS and SRE dataset, and ML-CRF performed the best for the RQA dataset. Error analysis showed that the BERT model performed well for DOORS data as it only missed a small percentage of the entities, and detected the boundary and entity types in partial and exact matching with high precision and recall. We also evaluated a few-shot NER model on the DOORS dataset, which was found to be a sample efficient technique, achieving better performance for small training sets.

Overall, fine-tuned transformer-based NER models showed promise to improve the process of software-specific entity extraction in generic SRS documents. These NER models can also benefit other downstream NLP tasks in the requirement engineering domain such as requirement classification, ambiguity and conflict detection in software requirements, and requirement quality assessment, which we aim to investigate as a future work. We also plan to explore other advanced few-shot learning methodologies, which provide opportunities to work with other software requirement datasets that have few labeled instances. Lastly, data

augmentation techniques can be employed to improve the support for minority entities in our datasets, and can help enhance overall NER performance.

References

[1] J. Lafferty, A. McCallum, and F. C. Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In: (2001).

[2] J. Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[3] Y. Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[4] Z. Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2020. arXiv: 1909.11942 [cs.CL].

[5] L. Cui et al. "Template-based named entity recognition using BART". In: *arXiv preprint arXiv:2106.01760* (2021).

[6] J. Tabassum et al. *Code and Named Entity Recognition in StackOverflow*. 2020. arXiv: 2005.01634 [cs.CL].

[7] M. Tikhomirov et al. "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.

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

[9] A. Nayak, V. Kesri, and R. K. Dubey. "Knowledge graph based automated generation of test cases in software engineering". In: *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*. 2020, pp. 289–295.

[10] D. Ye et al. "Software-specific named entity recognition in software engineering social content". In: *SANER'16*. Vol. 1. IEEE. 2016, pp. 90–101.

[11] M. V. P. Reddy et al. "NERSE: Named Entity Recognition in Software Engineering as a Service". In: *Service Research and Innovation*. Springer, 2018, pp. 65–80.

[12] Y. Yang and A. Katiyar. *Simple and Effective Few-Shot Named Entity Recognition with Structured Nearest Neighbor Learning*. 2020. arXiv: 2010.02405 [cs.CL].

[13] S. S. S. Das et al. *CONTaiNER: Few-Shot Named Entity Recognition via Contrastive Learning*. 2021. arXiv: 2109.07589 [cs.CL].

[14] J. Li et al. "Few-Shot Named Entity Recognition via Meta-Learning". In: *IEEE Transactions on Knowledge and Data Engineering* (2020), pp. 1–1. DOI: 10.1109/TKDE.2020.3038670.

[15] A. Fritzler, V. Logacheva, and M. Kretov. "Few-shot classification in named entity recognition task". In: *Proceedings of the ACM Symposium on Applied Computing*. 2019, pp. 993–1000.

[16] E. Hull, K. Jackson, and J. Dick. "DOORS: a tool to manage requirements". In: *Requirements engineering*. Springer, 2002, pp. 187–204.

[17] J. Cleland-Huang, M. Vierhauser, and S. Bayley. "Dronology: An incubator for cyber-physical system research". In: *arXiv preprint arXiv:1804.02423* (2018).

[18] A. Ferrari, G. O. Spagnolo, and S. Gnesi. "Pure: A dataset of public requirements documents". In: *RE'17*. IEEE. 2017, pp. 502–505.

[19] A. Post and T. Fuhr. "Case study: How Well Can IBM's" Requirements Quality Assistant" Review Automotive Requirements?" In: *REFSQ Workshops*. 2021.

[20] S. Friedenthal, R. Griego, and M. Sampson. "INCOSE model based systems engineering (MBSE) initiative". In: *INCOSE 2007 symposium*. Vol. 11. sn. 2007.

[21] J. Li et al. "A survey on deep learning for named entity recognition". In: *IEEE Transactions on Knowledge and Data Engineering* (2020).

[22] X. Yang et al. "Clinical concept extraction using transformers". In: *Journal of the American Medical Informatics Association* 27.12 (2020), pp. 1935–1942.

[23] N. UzZaman et al. "Semeval-2013 task 1: Tempeval-3: Evaluating time expressions, events, and temporal relations". In: *SemEval'13*. 2013, pp. 1–9.

[24] N. Chinchor and B. M. Sundheim. "MUC-5 evaluation metrics". In: *Proceedings of the 5th conference on message understanding*. 1993.