Code and Named Entity Recognition in StackOverflow

Jeniya Tabassum, Mounica Maddela, Wei Xu, Alan Ritter
Department of Computer Science and Engineering
The Ohio State University
{bintejafar.1, maddela.4, xu.1265, ritter.1492}@osu.edu

Abstract

There is an increasing interest in studying natural language and computer code together, as large corpora of programming texts become readily available on the Internet. For example, StackOverflow currently has over 15 million programming related questions written by 8.5 million users. Meanwhile, there is still a lack of fundamental NLP techniques for identifying code tokens or software-related named entities that appear within natural language sentences. In this paper, we introduce a new named entity recognition (NER) corpus for the computer programming domain, consisting of 15,372 sentences annotated with 20 fine-grained entity types. We also present the SoftNER model that combines contextual information with domain specific knowledge using an attention network. The code token recognizer combined with an entity segmentation model we proposed, consistently improves the performance of the named entity tagger. Our proposed SoftNER tagger outperforms the BiLSTM-CRF model with an absolute increase of +9.73 F_1 score on StackOverflow data.[1]

1 Introduction

Recently there has been significant interest in modeling human language together with computer code (Quirk et al., 2015; Iyer et al., 2016; Yin and Neubig, 2018), as more data becomes available on websites such as StackOverflow and GitHub. This is an ambitious yet promising direction for scaling up language understanding to richer domains. Access to domain-specific NLP tools could help a wide range of downstream applications. For example, extracting software knowledge bases from text (Movshovitz-Attias and Cohen, 2015), developing better quality measurements of StackOverflow posts (Ravi et al., 2014), finding similar questions (Amirreza Shirani, 2019) and more. However, there is a lack of NLP resources and techniques for identifying software-related named entities (e.g., variable names or application names) within natural language texts.

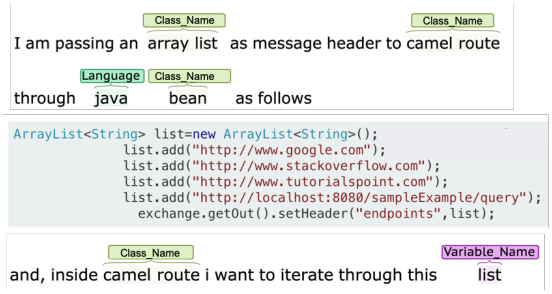

Figure 1: Examples of software-related named entities in a StackOverflow post.

In this paper, we present a comprehensive study that investigates the unique challenges of named entity recognition in the social computer programming domain. These named entities are often ambiguous and have implicit reliance on the accompanied code snippets. For example, the word '*list*' commonly refers to a data structure, but can also be used as a variable name (Figure 1). In order to recognize these entities, we propose an named entity recognizer (NER) that utilizes a multi-level attention network to combine the textual context with the code snippet knowledge. Using our newly annotated corpus of 15,372 sentences from StackOverflow, we rigorously test our proposed model which outperforms state-of-the-art BiLSTM-CRF tagging models for identifying 20 types of software-related named entities. Our key contributions are the following:

- A new StackOverflow NER corpus manually annotated with 20 types of named entities, including all in-line code within natural language sentences (§2). We demonstrate that NER in the software domain is an

[1]Our code and data are available at: https://github.com/jeniyat/StackOverflowNER/.

ideal benchmark task for testing effectiveness of contextual word representations, such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019), due to its inherent polysemy and salient reliance on context. For example, '*windows*' can be an English word, a variable, or a computer operating system, entirely depending on context.

- An in-domain trained neural NER tagger for StackOveflow (§3) that can recognize 20 fine-grained named entities related to software developing. We also tested its performance on GitHub text data, which include readme files and issue reports.
- A code token recognizer (§3.1) that utilizes StackOveflow code snippets to capture the character patterns of code related entities, and consistently improves the NER tagger.
- In-domain trained ELMo and BERT representations (§3.3) on 152 million sentences from StackOverflow that leads to more than 14 points increase in F_1 score over off-the-shelf ELMo, and significantly outperforms off-the-shelf BERT.

Our named entity tagger achieves a 78.41% F_1 score on StackOverflow and 62.69% F_1 score on GitHub data for extracting the 20 software related named entity types. We believe this performance is sufficiently strong to be practically useful. We have released our data and code, including the named entity tagger, our annotated corpus, annotation guideline, a specially designed tokenizer, and pre-trained StackOverflow ELMo and BERT embeddings.

2 Annotated StackOverflow Corpus

In this section, we describe the construction of our StackOverflow NER corpus. We randomly selected 1,237 question-answer threads from StackOverflow 10-year archive (from September 2008 to March 2018) and manually annotated them with 20 types of entities. For each question, four answers were annotated, including the accepted answer, the most upvoted answer, as well as two randomly selected answers (if they exist). Table 1 shows the statistics of our corpus. 40% of the question-answer threads were double-annotated, which are used as the development and test sets in our experiments (§4). We also annotated 6,501 sentences from GitHub readme files and issue reports as additional evaluation data.

	Train	Dev	Test	Total
#questions	741	247	249	1,237
#answers	897	289	315	1,501
#sentences	9,315	2,942	3,115	15,372
#tokens	136,996	43,296	45,541	225,833
#entities	11,440	3,949	3,733	19,122

	per Question	per Answer
avg. #sentences	6.84	4.60
avg. #tokens	98.46	69.37
avg. #entities	7.62	5.11
avg. #tokens per sentence	14.38	15.08

Table 1: Statistics of our StackOverflow NER corpus. These counts exclude all the code blocks and outputs blocks (i.e., lines that appear within ⟨code⟩ and ⟨blockquote⟩ tags).

2.1 Annotation Schema

We defined and annotated 20 types of fine-grained entities, including 8 code-related entities and 12 natural language entities. The code entities include mentions of CLASS, VARIABLE, IN LINE CODE, FUNCTION, LIBRARY, VALUE, DATA TYPE, and HTML XML TAG. Whereas the natural language entities include mentions of APPLICATION, UI ELEMENT, LANGUAGE, DATA STRUCTURE, ALGORITHM, FILE TYPE, FILE NAME, VERSION, DEVICE, OS, WEBSITE, and USER NAME.

Our annotation guideline was developed through several pilots and further updated with notes to resolve difficult cases as the annotation progressed.[2] Each entity type was defined to encourage maximum span length (e.g., '*SGML parser*' instead of '*SGML*'). We annotated noun phrases without including modifiers (e.g., '*C*' instead of '*Plain C*'), except a few special cases (e.g., '*rich text*' as a common FILE TYPE). On average, an entity contains about 1.5 tokens. While VARIABLE, FUNCTION and CLASS names mostly consist of only a single token, our annotators found that some are written as multiple tokens when mentioned in natural language text (e.g., '*array list*' for '*ArrayList*' in Figure 1). The annotators were asked to read relevant code blocks or software repositories to make a decision, if needed. Annotators also searched Google or Wikipedia to categorize unfamiliar cases.

The annotators were asked to update, correct, or add annotations from the user provided ⟨code⟩ markdown tags. StackOverflow users can utilize ⟨code⟩ markdowns to highlight the code entities within the natural language sentences. However,

[2]Our annotation guideline is available at: https://github.com/jeniyat/StackOverflowNER/.

in reality, many users do not enclose the code snippets within the ⟨code⟩ tags; and sometimes use them to highlight non-code elements, such as email addresses, user names, or natural language words. While creating the StackOverflow NER corpurs, we found that 59.73% of code-related entities are not marked by the StackOverflow users. Moreover, only 75.54% of the ⟨code⟩ enclosed texts are actually code-related, while 10.12% used to are highlighting natural language texts. The rest of cases are referring to non-code entities, such as SOFTWARE NAMES and VERSIONS. While markdown tag could be a useful feature for entity segmentation (§3.1.3), we emphasize the importance of having a human annotated corpus for training and evaluating NLP tools in the software domain.

2.2 Annotation Agreement

Our corpus was annotated by four annotators who are college students majored in computer science. We used a web-based annotation tool, BRAT (Stenetorp et al., 2012), and provided annotators with links to the original post on StackOverflow. For every iteration, each annotator was given 50 question-answer threads to annotate, 20 of which were double-annotated. An adjudicator then discussed disagreements with annotators, who also cross-checked the 30 single-annotated questions in each batch. The inter-annotator agreement is 0.62 before adjudication, measured by span-level Cohen's Kappa (Cohen, 1960).

2.3 Additional GitHub Data

To better understand the domain adaptability of our work, we further annotated the readme files and issue reports from 143 randomly sampled repositories in the GitHub dump (Gousios and Spinellis, 2012) (from October 29, 2007 to December 31, 2017). We removed all the code blocks from the issue reports and readme files collected from these 143 repositories. The resulting GitHub NER dataset consists of 6,510 sentences and 10,963 entities of 20 types labeled by two in-house annotators. The inter-annotator agreement of this dataset is 0.68, measured by span-level Cohen's Kappa.

2.4 StackOverflow/GitHub Tokenization

We designed a new tokenizer, SOTOKENIZER, specifically for the social computer programming domain. StackOverflow and GitHub posts exhibit common features of web texts, including abbreviations, emoticons, URLs, ungrammatical sentences and spelling errors. We found that tokenization is non-trivial as many code-related tokens are mistakenly split by the existing web-text tokenizers, including the CMU Twokenizer (Gimpel et al., 2011), Stanford TweetTokenizer (Manning et al., 2014), and NLTK Twitter Tokenizer (Bird et al., 2009):

txScope.Complete()	[*'txScope' '.' 'Complete' '(' ')'*]
std::condition_variable	[*'std' ':' ':' 'condition_variable'*]
math.h	[*'math' '.' 'h'*]
⟨span⟩	[*'⟨' 'span' '⟩'*]
a==b	[*'a' '=' '=' 'b'*]

Therefore, we implemented a new tokenizer, using Twokenizer[3] as the starting point and added additional regular expression rules to avoid splitting code-related tokens.

3 Named Entity Recognition

The extraction of software-related named entities imposes significant challenges as it requires resolving a significant amount of unseen tokens, inherent polysemy, and salient reliance on context. Unlike news or biomedical data, spelling patterns and long-distance dependencies are more crucial in the software domain to resolve ambiguities and categorize unseen words. Taken in isolation, many tokens are highly ambiguous and can refer to either programming concepts or common English words, such as '*go*', '*react*', '*spring*', '*while*', '*if*', '*select*'. Therefore, we design the SoftNER model that leverages sentential context to disambiguate and domain-specific character representations to handle rare words. Figure 2 shows the architecture of our model, which consists of primarily three components:

1. An **embedding extraction layer** (§3.1) that creates contextualized ELMo embeddings and two new domain-specific embeddings for each word in the input sentence.

2. A **multi-level attention layer** (§3.2) that combines the three word embeddings using an embedding-level and a word-level attention network.

3. A **BiLSTM-CRF** layer that predicts the entity type of each word using the weighted word representations from the previous layer.

[3] https://github.com/myleott/ark-twokenize-py

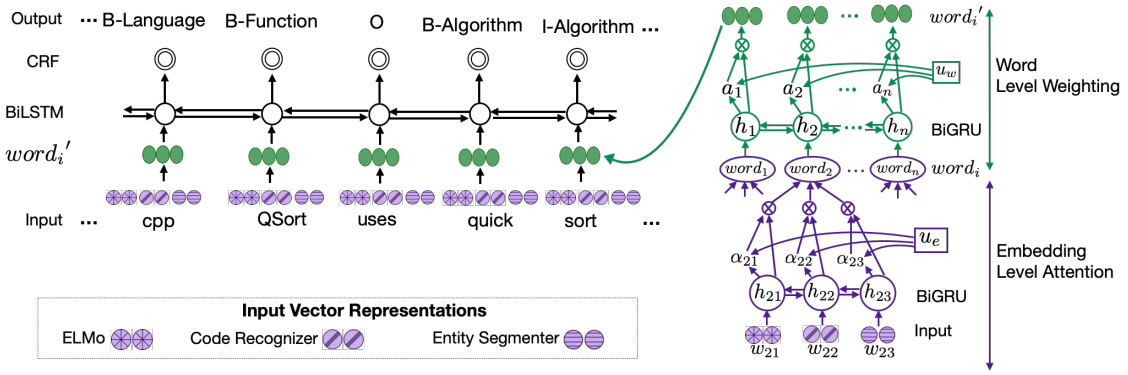

Figure 2: Our SoftNER model. It utilizes an attention network to combine the contextual word embeddings (ELMo) with the domain-specific embeddings (Code Recognizer and Entity Segmenter). The detailed structure of the attention network is depicted on the right.

3.1 Input Embeddings

For each word in the input sentence, we extract ELMo (Peters et al., 2018) representation and two new domain-specific embeddings produced by (i) a **Code Recognizer**, which represents if a word can be part of a code entity regardless of context; and (ii) an **Entity Segmenter**, that predicts whether a word is part of any named entity in the given sentence. Each domain-specific embedding is created by passing a binary value, predicted by a network independent from the SoftNER model, through an embedding layer. We describe the two standalone auxiliary models that generate these domain-based vectors below.

3.1.1 In-domain Word Embeddings

Texts in the software engineering domain contain programming language tokens, such as variable names or code segments, interspersed with natural language words. This makes input representations pre-trained on general newswire text unsuitable for software domain. Therefore, we pre-trained different in-domain word embeddings, including ELMo, BERT and GloVe vectors on the StackOverflow 10-year archive of 2.3 billion tokens (§3.3).

3.1.2 Context-independent Code Recognition

Humans with prior programming knowledge can easily recognize that '*list()*' is code, '*list*' can be either code or a verb, whereas '*listing*' is more likely a non-code token. We introduce a code recognition model to capture such prior probability of how likely a word can be a code token without considering any contextual information. It is worth noting that this standalone code recognition model is also useful for language-and-code research, such as retrieving code snippets based on natural language queries (Iyer et al., 2016; Giorgi and Bader, 2018; Yao et al., 2019)

Our code recognition model, which eventually generates the **Code Recognizer** vector, is a binary classifier. It utilizes language model features and character patterns to predict whether a word is a code entity. The input features include unigram word and 6-gram character probabilities from two language models (LMs) that are trained on the Gigaword corpus (Napoles et al., 2012) and all the code-snippets from the StackOverflow 10-year archive respectively. We also pre-trained FastText (Joulin et al., 2016) word embeddings using these code-snippets, where a word vector is represented as a sum of its character ngrams. We first transform each ngram probability into a k-dimensional vector using Gaussian binning (Maddela and Xu, 2018), which has shown to improve the performance of neural models using numeric features (Sil et al., 2017; Liu et al., 2016; Maddela and Xu, 2018). We then feed the vectorized features into a linear layer, concatenate the output with FastText character-level embeddings, and pass them through another hidden layer with *sigmoid* activation. We predict the token as a code-entity if the output probability is greater than 0.5.

3.1.3 Entity Segmentation

The segmentation task refers to identifying entity spans without assigning entity category. Segmentation is simpler and less error-prone than entity recognition as it does not require a fine-grained classification of the segmented tokens. In fact, a segmentation model trained on our annotated StackOverflow corpus achieves an accuracy of 97.4 on the dev set (details in §4.5). To leverage the high performance of segmentation for entity recognition, we introduce **Entity Segmenter**,

which predicts whether each token is an entity mention in the given sentence. For this binary tagging task, the model classifies a token as either I-ENTITY or O, instead of the traditional BIO scheme.

Our segmentation model, which generates the entity segmenter vector, consists of a BiLSTM encoder and a CRF decoder. For input, we concatenate ELMo embeddings with two hand-crafted features, namely **word frequency** and **code markdown**. Inclusion of hand-crafted features is influenced by Wu et al. (2018), where word-shapes and POS tags were shown to improve the performance of sequence tagging models.

Word Frequency represents the word occurrence count in the training set. As many code tokens are defined by individual users, they occur much less frequently than normal English words. In fact, code and non-code tokens have an average frequency of 1.47 and 7.41 respectively in our corpus. Moreover, ambiguous token that can be either code or non-code entities have a much higher average frequency of 92.57. To leverage this observation, we include word frequency as a feature, converting the scalar value into a k-dimensional vector by Gaussian binning (Maddela and Xu, 2018).

Code Markdown indicates whether the given token appears inside a ⟨code⟩ markdown tag in the StackOverflow post. It is worth noting that ⟨code⟩ tags are noisy as users do not always enclose inline code in a ⟨code⟩ tag or use the tag to highlight non-code texts (details in §2.1). However, we find it helpful to include the markdown information as a feature as it improves the performance of our segmentation model.

3.2 Multi-Level Attention

We build an aggregated word vector from the input embeddings using a multi-level attention network similar to Yang et al.(2016). We combine the input embeddings in the first attention layer and calculate the importance of each word for the task in the second layer. Although such embedding-level attention is not commonly used in NER, we found it empirically helpful for the software domain.

Embedding-Level Attention We use three embeddings, ELMo (w_{i1}), Code Recognizer (w_{i2}), and Entity Segmenter (w_{i3}), for each word w_i in the input sentence. We introduce the embedding-level attention α_{it} ($t \in \{1, 2, 3\}$) to capture each embedding's contribution towards the meaning of the word. To compute α_{it}, we pass the input embeddings through a bidirectional GRU and generate their corresponding hidden representations $h_{it} = \overleftrightarrow{GRU}(w_{it})$. These vectors are then passed through a non-linear layer, which outputs $u_{it} = tanh(W_e h_{it} + b_e)$. We introduce an embedding-level context vector, u_e, which is learned during the training process. This context vector is combined with the hidden embedding representation using a softmax function to extract weight of the embeddings, $\alpha_{it} = \frac{exp({u_{it}}^T u_e)}{\sum_t exp({u_{it}}^T u_e)}$. Finally, we create the word vector by a weighted sum of all the information from different embeddings as $word_i = \sum_t \alpha_{it} h_{it}$.

Weighted Word Representation We also use a word-level weighting factor α_i to emphasize the importance of each word w_i for the NER task. Similar to the embedding-level attention, we calculate α_i from the weighted word vectors $word_i$. We use bidirectional GRU to encode the summarized information from neighbouring words and get $h_i = \overleftrightarrow{GRU}(word_i)$. This is then passed through a hidden layer which outputs $u_i = tanh(W_w h_i + b_w)$. Using this vector, we extract the normalized weight for each word vector $\alpha_i = \frac{exp({u_i}^T u_w)}{\sum_t exp({u_i}^T u_w)}$, where u_w is another word-level context vector that is learned during training. Finally, we compute the weighted word representation $word'_i = \alpha_i h_i$. The aggregated word vector $word'_i$ is then fed into a BiLSTM-CRF network, which predicts the entity category for each word.

3.3 Implementation Details

We use PyTorch framework to implement our proposed SoftNER model and two auxiliary systems, namely the code recognition and the entity segmentation systems. Our SoftNER model consists of a BiLSTM encoder with character-level CNN features and a CRF decoder. The input of the network consists of 500-dimensional segmenter vectors, 300-dimensional code recognizer vectors and 1024-dimensional contextual word representations. To extract in-domain word representations, we pre-trained Glove, ELMo and BERT vectors on 152 million sentences from the StackOverflow archive, excluding all the sentences from the 1,237 posts in our annotated corpus. The pre-training of 300-dimensional Glove embeddings, with a frequency cut-off of 5, took 8 hours on

	P	R	F_1
Test set			
Feature-based CRF	71.77	39.70	51.12
Fine-tuned BERT	45.92	77.02	57.54
Fine-tuned BERTOverflow	68.77	67.47	68.12
BiLSTM-CRF (ELMoVerflow)	73.03	64.82	68.68
SoftNER (ELMoVerflow)	**78.22**	**78.59**	**78.41**
Dev set			
Feature-based CRF	66.85	46.19	54.64
Fine-tuned BERT	46.42	79.57	58.64
Fine-tuned BERTOverflow	72.11	70.51	71.30
BiLSTM-CRF (ELMoVerflow)	74.44	68.71	71.46
SoftNER (ELMoVerflow)	**79.43**	**80.00**	**79.72**

Table 2: Evaluation on the *dev* and *test* sets of the StackOverflow NER corpus. Our SoftNER model outperforms the existing approaches.

a server with 32 CPU cores and 386 GB memory. The pre-training of 1024-dimensional ELMo vectors took 46 days on 3 NVIDIA Titan X Pascal GPUs. Pre-training of the BERT$_{base}$ model, with 64,000 WordPiece vocabulary, took 7 days on Google TPU.

We train the SoftNER model and the two auxiliary systems separately. Our segmentation model follows the same architecture and training setup as SoftNER except for the input, where ELMo embeddings are concatenated with 100-dimensional code markdown and 10-dimensional word frequency features. We set the number of bins k to 10 for Gaussian vectorization. Our code recognition model is a feedforward network with two hidden layers and a single output node with $sigmoid$ activation.

4 Evaluation

In this section, we show that our SoftNER model outperforms all the previous NER approaches on the StackOverflow and GitHub data. We also discuss the factors pivotal to the performance of our model, namely pre-trained in-domain ELMo embeddings and our two domain-specific vectors.

4.1 Data

We train and evaluate our SoftNER model on the StackOverflow NER corpus of 9,352 train, 2,942 development and 3,115 test sentences we constructed in §2. We use the same data for our segmentation model but replace all the entity tags with an I-ENTITY tag. For the code recognition model, we created a lexicon of 6000 unique words randomly selected from the *train* set of the StackOverflow NER corpus. Each word was labelled individually without context as CODE, AMBIGUOUS or NON-CODE by two annotators. The inter-annotator agreement was 0.89, measured by Cohen's Kappa. After discarding disagreements, we divided the remaining 5312 tokens into 4312 train and 1000 test instances. Then, we merged AMBIGUOUS and NON-CODE categories to facilitate binary classification. We name this dataset of 5312 individual tokens as SOLEXICON.

4.2 Baselines

We compare our model with the following baseline and state-of-the-art approaches:

- A **BiLSTM-CRF** model with in-domain ELMo embeddings (**ELMoVerflow**; details in §3.3). This architecture is the state-of-the-art baseline NER models in various domains (Lample et al., 2016; Kulkarni et al., 2018; Dai et al., 2019).
- A **Fine-tuned in-domain BERT** model where we fine-tune the in-domain pre-trained-BERT$_{base}$ cased (**BERTOverflow**; details in §3.3) checkpoint[4] with our annotated corpus.
- A **Fine-tuned out-domain BERT** model where we fine-tune the out-domain BERT$_{base}$ cased checkpoint[5] with our annotated corpus.
- A **Feature-based Linear CRF** model which uses the standard orthographic, context and gazetteer features, along with the code markdown tags and handcrafted regular expressions to recognize code entities (details in Appendix A).

4.3 Results

Table 2 shows the precision (**P**), recall (**R**) and $\mathbf{F}_1$ score comparison of different models evaluated on the StackOverflow NER corpus. Our SoftNER model outperforms the existing NER approaches in all the three metrics. Compared to BiLSTM-CRF, SoftNER demonstrates a 9.7 increase in F_1 on the *test* set.

4.4 In-domain vs. Out-domain Word Embeddings

Table 3 shows the performance comparison between in-domain and out-domain word embeddings. We consider off-the-shelf ELMo (Peters

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

[5] https://github.com/google-research/BERT

et al., 2018) and GloVe (Pennington et al., 2014) vectors trained on newswire and web texts as out-domain embeddings. Using the state-of-the-art BiLSTM-CRF model (Lample et al., 2016; Kulkarni et al., 2018; Dai et al., 2019), we observe a large increase of 13.64 F_1 score when employing in-domain ELMo (**ELMoVerflow**) representations over in-domain GloVe (**GloVeOverflow**), and an increase of 15.71 F_1 score over out-domain ELMo. We found that fine-tuned out-domain BERT (Devlin et al., 2019) outperforms the out-domain ELMo (Table 3), although it under-performs in-domain ELMo (ELMoVerflow) by 12.8 F_1 score (Table 2) on our StackOverflow NER corpus. Similarly, for Github data (more details in §5), in-domain ELMo outperforms the out-domain fine-tuned BERT by 10.67 F_1 score (Table 8). In our experiments, fine-tuned BERTOverflow extracts the named entities with higher recall, whereas the ELMoVerflow extracts them with higher precision. However, as the overall F_1 score of the ELMoVerflow is slightly higher than the BERTOverflow, we used in-domain ELMo for the rest of our experiments.

It is worth noting that, the performance improvements from contextual word embeddings are more pronounced on our software domain than on newswire and biomedical domains. Original ELMo and BERT outperform GloVe by 2.06 and 2.12 points in F_1 respectively on CoNLL 2003 NER task of newswire data (Peters et al., 2018; Devlin et al., 2019). For biomedical domain, in-domain ELMo outperforms out-domain ELMo by 1.33 points in F_1 on the BC2GM dataset (Sheikhshabbafghi et al., 2018).

We hypothesized that the performance gains from the in-domain contextual embeddings are largely aided by the model's ability to handle ambiguous and unseen tokens. The increase in performance is especially notable (41% → 70% accuracy) for unseen tokens, which constitute 38% of the tokens inside gold entity spans in our dataset. This experiment also demonstrates that our annotated NER corpus provides an attractive test-bed for measuring the adaptability of different contextual word representations.

4.5 Evaluation of Auxiliary Systems

Our domain-specific vectors, namely **Code Recognizer** and **Entity Segmenter** are also crucial for the overall performance of our SoftNER model.

	P	R	F_1
out-domain Word Embeddings			
GloVe (newswire+Wiki+Web)	61.71	49.08	54.67
ELMo (newswire+Wiki)	67.66	47.41	55.75
Fine-tuned BERT (book+Wiki)	45.92	77.02	57.54
In-Domain Word Embeddings (trained on StackOverflow)			
GloVeOverflow	66.28	51.28	57.82
ELMoVerflow	**74.44**	68.71	**71.46**
Fine-tuned BERTOverflow	72.11	**70.51**	71.30

Table 3: Performance of fine-tuned BERT and BiLSTM-CRF model with different input representations on the *dev* set of our StackOverflow NER corpus. Contextualized word representations show a clear benefit relative to GloVe, when trained on StackOverflow data.

	P	R	F_1
Token Frequency	33.33	2.25	4.22
Most Frequent Label	**82.21**	58.59	68.42
Our Code Recognition Model	**78.43**	83.33	**80.80**
– Character ngram LMs	64.13	**84.51**	72.90
– Word ngram LMs	67.98	72.96	70.38
– FastText Embeddings	76.12	81.69	78.81

Table 4: Evaluation results and feature ablation of our code recognition model on SOLEXICON *test* set of 1000 manually labeled unique tokens, which are sampled from the *train* set of StackOverflow NER corpus.

Table 6 shows an ablation study. Removing code recognizer vectors and segmenter results in a 2.25 and 4.65 drops in F_1 scores respectively. If we replace embedding-level attention with a simple concatenation of embeddings, the performance also drop by 1.84 in F_1. In addition, we evaluate the effectiveness of our two domain-specific auxiliary systems on their respective tasks:

Code Recognition Table 4 compares the performance of our code recognition model with other baselines on the SLEXICON *test* set (§4.1), which consists of 1000 random words from the *train* set of StackOverflow NER corpus classified as either a code or a non-code token. The baselines include: (i) a Most Frequent Label baseline, which assigns the most frequent label according to the human annotation in SOLEXICON *train* set; and (ii) a frequency baseline, which learns a threshold over token frequency in the *train* set of StackOverflow NER corpus using a decision tree classifier. Our model outperforms both baselines in terms of F_1 score. Although the most frequent label baseline achieves better precision than our model, it performs poorly on unseen tokens resulting in a large drop in recall and F_1 score. The ablation experiments show that the FastText word embeddings along with the character and word-level features are crucial for the code token recognition task.

	P	R	F_1
Stanford NER Tagger	63.02	5.74	10.52
Our Entity Segmentation Model	**86.80**	**81.86**	**84.26**
– Word Frequency	84.61	81.53	83.04
– Code Markdown	82.49	81.83	82.16

Table 5: Evaluation of our segmentation model on the *dev* set of the StackOverflow NER corpus.

	P	R	F_1
SoftNER	**79.43**	**80.00**	**79.72**
– Multi-level Attention	77.68	78.08	77.88
– Code Recognizer	77.18	77.76	77.47
– Entity Segmenter	74.82	75.32	75.07

Table 6: Ablation study of SoftNER on the *dev* set of StackOverflow NER corpus.

Entity Segmentation Table 5 shows the performance of our segmentation model on the *dev* set of our StackOverflow corpus, where the entity tags are replaced by an I-ENTITY tag. Our model achieves an F_1 score of 84.3 and an accuracy of 97.4. Incorporating word frequency and code markdown feature increases the F_1 score by 1.2 and 2.1 points respectively. The low 10.5 F_1 score of Stanford NER tagger (Manning et al., 2014), which is trained on newswire text, demonstrates the importance of domain-specific tools for the software engineering domain.

4.6 Error Analysis

Based on our manual inspection, the incorrect predictions made by NER systems can be largely classified into the following two categories (see examples in Table 7):

Segmentation Mismatch	**Gold:** installing on Windows Server 2012 R2. **Predicted:** installing on Windows Server 2012 R2.
Entity-Type Mismatch	**Gold:** I used ShareKit which was a breeze to add. **Predicted:** I used ShareKit which was a breeze to add.

Table 7: Representative examples of error categories. In each example pair, the first sentence contains the gold entities, and the second sentence contains the predicted entities from NER model.

Segmentation Mismatch refers to the cases where model predicts the boundary of entities incorrectly. Our SoftNER model reduces such segmentation errors by 80.33% compared to the BiLSTM-CRF baseline.

Entity-Type Mismatch refers to the errors where a code entity (e.g., names of variables) is predicted as a non-code entity (e.g., names of devices), and vice-versa. Our SoftNER model reduces such entity type errors by 23.34% compared to the BiLSTM-CRF baseline.

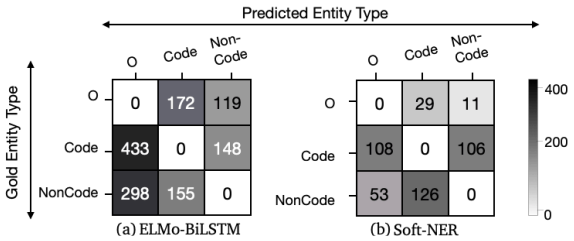

Figure 3: Comparison of errors made by the ELMo BiLSTM-CRF baseline and our SoftNER on the *dev* set of StackOverflow NER corpus. In the error heatmap, darker cell color corresponds to higher error counts. Our SoftNER model reduces errors in all the categories.

	P	R	F_1
Feature-Based CRF	43.16	35.71	39.09
Fine-tuned out-domain BERT	56.59	48.13	52.02
Fine-tuned BERTOverflow	61.71	58.75	60.19
BiLSTM-CRF (ELMoGitHub)	**64.53**	**60.96**	**62.69**
SoftNER (ELMoVerflow)	62.05	59.20	60.59
SoftNER (ELMoGitHub)	63.29	60.89	62.07

Table 8: Evaluation on the GitHub NER dataset of readme files and issue posts. *All the models are trained on our StackOverflow NER corpus.* Our SoftNER model performs close to BiLSTM-CRF model trained on the GitHub ELMo embeddings.

As illustrated in Figure 3, our SoftNER model reduced the errors in both categories by incorporating the auxilary output from segmenter and code recognizer model.

5 Domain Adaptation to GitHub data

To understand the domain adaptability of our StackOverflow based SoftNER, we evaluate its performance on readme files and issue reports from 143 randomly sampled repositories in the GitHub dump (Gousios and Spinellis, 2012). We also trained GitHub ELMo embeddings on 4 million sentences from randomly sampled 5,000 GitHub repositories.

Table 8 shows that the performance of our SoftNER model using StackOverflow ELMo embeddings is similar to the top performing BiLSTM-CRF model using GitHub ELMo embeddings with a difference of only 2.1 points in F_1. We also did not observe any significant gain after adding the code recognizer and segmenter vectors to the Github ELMo embeddings. We think one likely explanation is that GitHub data contains less code-related tokens when compared to StackOverflow. The percentage of code-related entity tokens is 63.20% in GitHub and 77.21% in StackOverflow.

Overall, we observe a drop of our SoftNER tagger from 78.41 on StackOverflow (Table 2) to 62.69 on GitHub data (Table 8) in F_1 due to domain mismatch. However, we believe that our NER tagger still achieves sufficient performance to be useful for applications on GitHub.[6] We leave further investigation of semi-supervised learning and other domain adaptation approach for future work.

6 Related Work

The CoNLL 2003 dataset (Sang and De Meulder, 2003) is a widely used benchmark for named entity recognition, which contains annotated newswire text from the Reuters RCV1 corpus. State-of-the-art approaches on this dataset (Baevski et al., 2019) use a bidirectional LSTM (Lample et al., 2016; Ma and Hovy, 2016) with conditional random field (Collobert et al., 2011) and contextualized word representations (McCann et al., 2017; Peters et al., 2018; Devlin et al., 2019).

Named entity recognition has been explored for new domains and languages, such as social media (Finin et al., 2010; Ritter et al., 2011; Plank et al., 2014; Derczynski et al., 2015; Limsopatham and Collier, 2016; Aguilar et al., 2017), biomedical texts (Collier and Kim, 2004; Greenberg et al., 2018; Kulkarni et al., 2018), multilingual texts (Benajiba et al., 2008; Xie et al., 2018) and code-switched corpora (Aguilar et al., 2018; Ball and Garrette, 2018). Various methods have been investigated for handling rare entities, for example incorporating external context (Long et al., 2017) or approaches that make use of distant supervision (Choi et al., 2018; Yang et al., 2018; Onoe and Durrett, 2019).

There has been relatively little prior work on named entity recognition in the software engineering domain. Ye et al. (2016) annotated 4,646 sentences from StackOverflow with five named entity types (Programming Language, Platform, API, Tool-Library-Framework and Software Standard). The authors used a traditional feature-based CRF to recognize these entities. In contrast, we present a much larger annotated corpus consisting of 15,372 sentences labeled with 20 fine-grained entity types. We also develop a novel attention based neural NER model to extract those fine-grained entities.

[6] As a reference, the state-of-the-art performance for 10-class Twitter NER is 70.69 F_1 (Zhang et al., 2018).

7 Conclusion

In this work, we investigated the task of named entity recognition in the social computer programming domain. We developed a new NER corpus, consisting of 15,372 sentences from StackOverflow and 6,510 sentences from GitHub annotated with 20 fine-grained named entities. We demonstrate that this new corpus is an ideal benchmark dataset for contextual word representations, as there are many challenging ambiguities that often require long-distance context to resolve. We proposed a novel attention based model, named SoftNER, that outperforms the state-of-the-art NER models on this dataset. Furthermore, we investigated the important sub-task of code recognition. Our novel code recognition model captures additional spelling information beyond character-based ELMo and consistently improves performance of the NER model. We believe our corpus and StackOverflow-based named entity tagger will be useful for various language-and-code tasks, such as code retrieval, software knowledge base extraction and automated question-answering.

Acknowledgement

We thank anonymous reviewers for their thoughtful comments. We also thank NVIDIA, Google, and Ohio Supercomputer Center (Center, 2012) for providing GPU/TPU computing resources; Wuwei Lan for kindly helping to train in-domain BERT on StackOverflow data; Sydney Lee, Rita Tong, Lillian Chow, and Raleigh Potluri for help with data annotation. This research is supported in part by the NSF awards IIS-1822754 and IIS-1845670, ODNI and IARPA via the BETTER program contract 19051600004, ARO and DARPA via the SocialSim program contract W911NF-17-C-0095, Criteo Faculty Research Award to Wei Xu, and Amazon Faculty Research Award to Alan Ritter. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of NSF, ODNI, IARPA, ARO, DARPA or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

Gustavo Aguilar, Fahad AlGhamdi, Victor Soto, Mona Diab, Julia Hirschberg, and Thamar Solorio. 2018. Named Entity Recognition on Code-Switched Data: Overview of the CALCS 2018 Shared Task. In *Proceedings of the Third Workshop on Computational Approaches to Linguistic Code-Switching*.

Gustavo Aguilar, Suraj Maharjan, Adrian Pastor López-Monroy, and Thamar Solorio. 2017. A Multi-task Approach for Named Entity Recognition in Social Media Data. In *Proceedings of the 3rd Workshop on Noisy User-generated Text (WNUT)*.

David Lo Thamar Solorio Amin Alipour Amirreza Shirani, Bowen Xu. 2019. Question Relatedness on Stack Overflow: The Task, Dataset, and Corpus-inspired Models. In *Proceedings of the AAAI Reasoning for Complex Question Answering Workshop*.

Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. 2019. Cloze-driven pretraining of self-attention networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics.

Kelsey Ball and Dan Garrette. 2018. Part-of-Speech Tagging for Code-Switched, Transliterated Texts without Explicit Language Identification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yassine Benajiba, Mona Diab, and Paolo Rosso. 2008. Arabic Named Entity Recognition using Optimized Feature Sets. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Ohio Supercomputer Center. 2012. Oakley supercomputer.

Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-Fine Entity Typing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*.

Nigel Collier and Jin-Dong Kim. 2004. Introduction to the Bio-entity Recognition Task at JNLPBA. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP)*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (almost) from Scratch. *Journal of Machine Learning Research (JMLR)*.

Xiang Dai, Sarvnaz Karimi, Ben Hachey, and Cecile Paris. 2019. Using Similarity Measures to Select Pretraining Data for NER. *arXiv preprint arXiv:1904.00585*.

Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke Van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, and Kalina Bontcheva. 2015. Analysis of Named Entity Recognition and Linking for Tweets. *Information Processing & Management*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Tim Finin, Will Murnane, Anand Karandikar, Nicholas Keller, Justin Martineau, and Mark Dredze. 2010. Annotating Named Entities in Twitter Data with Crowdsourcing. In *Proceedings of the Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-speech Tagging for Twitter: Annotation, Features, and Experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

John M Giorgi and Gary Bader. 2018. Transfer Learning for Biomedical Named Entity Recognition with Neural Networks. *bioRxiv*.

G. Gousios and D. Spinellis. 2012. GHTorrent: Github's Data from a Firehose. In *Proceedings of the 9th IEEEConference on Mining Software Repositories (MSR)*.

Nathan Greenberg, Trapit Bansal, Patrick Verga, and Andrew McCallum. 2018. Marginal Likelihood Training of BiLSTM-CRF for Biomedical Named Entity Recognition from Disjoint Label Sets. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing Source Code using a Neural Attention Model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. FastText.zip: Compressing Text Classification Models. *arXiv preprint arXiv:1612.03651*.

Chaitanya Kulkarni, Wei Xu, Alan Ritter, and Raghu Machiraju. 2018. An Annotated Corpus for Machine Reading of Instructions in Wet Lab Protocols. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Nut Limsopatham and Nigel Collier. 2016. Bidirectional LSTM for Named Entity Recognition in Twitter Messages. In *Proceedings of 2016 the Workshop on Noisy User-generated Text (WNUT)*.

Dan Liu, Wei Lin, Shiliang Zhang, Si Wei, and Hui Jiang. 2016. Neural Networks Models for Entity Discovery and Linking. *arXiv preprint arXiv:1611.03558*.

Teng Long, Emmanuel Bengio, Ryan Lowe, Jackie Chi Kit Cheung, and Doina Precup. 2017. World Knowledge for Reading Comprehension: Rare Entity Prediction with Hierarchical LSTMs Using External Descriptions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Mounica Maddela and Wei Xu. 2018. A Word-Complexity Lexicon and A Neural Readability Ranking Model for Lexical Simplification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the 2014 Association for Computational Linguistics System Demonstrations (ACL)*.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in Translation: Contextualized Word Vectors. In *Proceedings of the 2017 Conference on Neural Information Processing Systems (NeurIPS)*.

Dana Movshovitz-Attias and William W Cohen. 2015. KB-LDA: Jointly Learning a Knowledge Base of Hierarchy, Relations, and Facts. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*.

Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. 2012. Annotated Gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-Scale Knowledge Extraction*.

Yasumasa Onoe and Greg Durrett. 2019. Learning to Denoise Distantly-Labeled Data for Entity Typing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NACL)*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the of Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Barbara Plank, Dirk Hovy, and Anders Søgaard. 2014. Learning part-of-speech taggers with inter-annotator agreement loss. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to Code: Learning Semantic Parsers for If-This-Then-That Recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*.

Sujith Ravi, Bo Pang, Vibhor Rastogi, and Ravi Kumar. 2014. Great Question! Question Quality in Community Q&A. In *Eighth International AAAI Conference on Weblogs and Social Media*.

Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. 2011. Named Entity Recognition in Tweets: An Experimental Study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL*.

Golnar Sheikhshabbafghi, Inanc Birol, and Anoop Sarkar. 2018. In-domain Context-aware Token Embeddings Improve Biomedical Named Entity Recognition. In *Proceedings of the Ninth International Workshop on Health Text Mining and Information Analysis*.

Avirup Sil, Gourab Kundu, Radu Florian, and Wael Hamza. 2017. Neural Cross-Lingual Entity Linking. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. brat: a Web-based Tool for NLP-Assisted Text Annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Minghao Wu, Fei Liu, and Trevor Cohn. 2018. Evaluating the Utility of Hand-crafted Features in Sequence Labelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jiateng Xie, Zhilin Yang, Graham Neubig, Noah A Smith, and Jaime Carbonell. 2018. Neural Cross-Lingual Named Entity Recognition with Minimal Resources. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yaosheng Yang, Wenliang Chen, Zhenghua Li, Zhengqiu He, and Min Zhang. 2018. Distantly Supervised NER with Partial Annotation Learning and Reinforcement Learning. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*.

Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. 2019. CoaCor: Code Annotation for Code Retrieval with Reinforcement Learning. In *Proceedings of the World Wide Web Conference (WWW)*.

Deheng Ye, Zhenchang Xing, Chee Yong Foo, Zi Qun Ang, Jing Li, and Nachiket Kapre. 2016. Software-specific Named Entity Recognition in Software Engineering Social Content. In *Proceedings of the 2016 IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*.

Qi Zhang, Jinlan Fu, Xiaoyu Liu, and Xuanjing Huang. 2018. Adaptive Co-attention Network for Named Entity Recognition in Tweets. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

A Feature-Based CRF Baseline

We implemented a CRF baseline model using CRFsuite[7] to extract the software entities. This model uses standard orthographic, contextual and gazetteer features. It also includes the code markdown tags (§3.1.3) and a set of regular expression features. The regular expressions are developed to recognize specific categories of code-related entities. Feature ablation experiments on this CRF model are presented in Table 9. One noticeable distinction from the named entity recognizer in many other domains is that the *contextual features are not as helpful in feature-based CRFs for classifying software entities*. This is because, in the StackOverflow NER corpus a significant number of neighbouring words are shared among different software entities. As an example, the bigram *'in the'* frequently appears as the left context of the following types: APPLICATION, CLASS, FUNCTION, FILE TYPE, UI ELEMENT, LIBRARY, DATA STRUCTURE and LANGUAGE.

	P	**R**	**F_1**
Feature-based CRF	66.85	46.19	54.64
– Context Features	68.91	43.58	53.39
– Markdown Feature	70.64	40.15	51.20
– Rule and Gazetteer Features	69.71	40.66	51.36

Table 9: Feature based CRF performance with varying input features on *dev* data.

[7] http://www.chokkan.org/software/crfsuite/