Helpful or Not? An investigation on the feasibility of identifier splitting via CNN-BiLSTM-CRF

Jiechu Li*,† Qingfeng Du*†, Kun Shi*†§, Yu He*† Xin Wang*‡ and Jincheng Xu*†
*School of Software Engineering, Tongji University
†Software Engineering R&D Centre, Jishi Building, Tongji University
‡Smart City Labotary, Jishi Building, Tongji University
§Shanghai Research and Development Center, Baidu Inc.
Email: {lijiechu, Du_cloud, skyline, rainlf, wangxin16, xujincheng}@tongji.edu.cn

***Abstract*—We recently introduced a new technique to handle source code identifier splitting. The proposed technique, denoted as CNN-BiLSTM-CRF[a neural network composed of a convolutional neural network(CNN), bidirectional long short-term memory networks(BiLSTM) and conditional random fields(CRFs)] enables us to obtain a model that splits identifiers correctly and effectively. This technique combines the use of a CNN layer with the mature BiLSTM-CRF model. The experimental results indicate that CNN-BiLSTM-CRF delivers outstanding performance on all four of the evaluation oracles. More importantly, we endeavored to provide insight into the practical feasibility of this technique by considering the aspects of generality, data size in demand and construction cost, etc. Finally, we reasoned out that CNN-BiLSTM-CRF should be helpful and improvable for identifier splitting in practical works in terms of the accuracy and feasibility. This was validated by multifaceted experiments.**

***Index Terms*—identifier splitting, source code mining, program comprehension, CNN, BiLSTM-CRF, feasibility investigation**

I. Introduction

The rapid development of natural language processing (NLP) and machine learning has derived many prominent techniques[e.g., information retrieval(IR) model and deep learning] to support software engineering (SE) tasks[1], [2], [3], [4](e.g., feature location and traceability link recovery). These techniques extract domain concepts of corresponding software projects by analyzing textual information mined from software repositories.

Source code identifiers[1], occupying most of the characters[2] insides programs[6], are one of the critical components that can be mined from software artifacts. To our knowledge, this is owing to the fact that strong consistency is always reflected between the source code and other software artifacts (e.g., documentation). However, the definitions of identifiers are strictly constrained by the syntax rules of programming languages, complicating not only the manual recognition process but also the automatic NLP tokenization process on them.

[1]Source code identifiers are tokens that name language entities including variables, types, functions, and packages in programming languages. These tokens represent different meanings, according to their naming purposes.

[2]An identifier *id* is normally in the form of (s_0, s_1, s_2, ..., s_n), where s_i is a letter, digit, or special character[5].

DOI reference number: 10.18293/SEKE2018-167

To be specific, an appointed source code identifier must be composed of a series of terms without explicit blanks (e.g., *XMLParser1*, *dorapntr*, and *treeNode*) in which the terms are normal dictionary words (e.g., *Parser*) and acronyms (e.g., *XML*), abbreviations or, even worse, unmeaningful vocabs. Directly performing NLP procedures (e.g., word embedding) on these irregular strings inevitably degrades the performance of ongoing or upcoming concept-comprehension tasks. Hence, normalizing identifiers into their constituent parts is crucial when leveraging NLP techniques on SE tasks.

Identifier splitting is the first and highly critical step of identifier normalization, with the subsequent step of mapping or expanding the correct split terms into their original dictionary words[7], [8]. Identifier splitting is also considered as an indispensable procedure after tokenization[9], [10]. In fact, existing techniques have already been put forward with the aim to efficiently and correctly split identifiers, including the investigated approach in our study: identifier splitting via a trained **CNN-BiLSTM-CRF** model(CNN-BiLSTM-CRF is used to represent this approach itself in the following literature for short).

The mentioned CNN-BiLSTM-CRF, is derived from our early research achievements. It is proposed under the motivation for better identifier splitting techniques that better contribute to related SE tasks[11]. Our pioneering research and initial experimental results showed that the CNN-BiLSTM-CRF significantly outperforms other state-of-the-art techniques. However, it remains certain thoughtlessness before we resolve to apply this technique to split identifiers in practical SE tasks. These issues certainly exist and are determined by the methodology of CNN-BiLSTM-CRF, which takes samples of manually built oracles as input for training, then tunes and evaluates the outcome model with the rest of the oracle samples. In terms of practical feasibility, there are concerns regarding the **generalization capability** (refers to the ability to correctly split identifiers outside of the oracle with which it was trained) and the **training cost** of the model in addition to the superficial **outstanding performance**.

In this study, our specially designed experiments and qualitative analysis on the obtained results lead to the conclusion that splitting identifiers via CNN-BiLSTM-CRF is helpful and absolutely feasible in terms of the above indicated facets.

In summary, we make the following contributions:

- We prospectively present a new and novel identifier splitting method called CNN-BiLSTM-CRF, which possesses great ability to split source code identifiers accurately.
- We systematically investigate and inspect the practicability of the CNN-BiLSTM-CRF approach regarding its comprehensive aspects.
- We offer evidence that the investigated approach is not only superior to other state-of-the-art identifier splitting techniques but also of enough generality and feasibility based on empirical evaluations performed on a maximum of four benchmark oracles.
- We provide the publicly available package and implemented code for researchers to adopt, replicate, or further explore our work.

The remainder of this paper is structured as follows: Section II presents some significant related work on identifier splitting and concisely describes the mechanism of our proposed CNN-BiLSTM-CRF approach. Section III demonstrates the process of setting up our experiments and discusses the results. Section IV is a deeper discussion of our achieved results and of threats to the validity of this method. Section V concludes this study.

II. Background and Related Work

A. Previous Work on Identifier Splitting

Ideally, identifiers in recognizable forms such as the well-known *CamelCase* and *UnderscoreCase* can be effortlessly split by explicit rules (splitting at underscores or changes from lower case to upper case). However, extra strategies still need to be considered to effectively solve the splittings on identifiers in more sophisticated forms (e.g., all characters are in single case or containing digits). In this case, researchers are spurred to promote the performance of identifier splitting, and hence many innovative techniques have been introduced(e.g., [5], [12], [13], [14], [15] and etc.).

Here, we selectively and briefly introduce several of these state-of-the-art identifier splitting techniques. Field et al.[12] first attempted to split identifiers with a simple **greedy** algorithm and used a **simple artificial neural network** approach. **DTW**[14] is based on a modified version of *Dynamic Time Warping*. The fundamental idea behind this approach is to determine the optimal matching of two series of characters $(x_1, x_2, ..., x_{n)}$ and $(y_1, y_2, ..., y_m)$. **GenTest**[7] attempts to score all possible splits for an identifier with a series of *handcrafted metrics*, and the one with the highest score is considered to be the correct split. **LINSEN**[15] uses an efficient approximate string-matching algorithm, *BYP*, in conjunction with nested context-based dictionaries.

A number of man-made oracles for identifier splitting were created from past accomplishments. Thereupon then previous studies could evaluate their techniques on these available oracles. In our study, four frequently used oracles namely *Binkley*[16](containing 2,663 samples), *BT11*[17](21,122), *Jhotdraw*(974), and *Lynx*(3,085)[14], will be applied.

B. Reformulating Identifier Splitting as a Sequence Labeling Task

The essence of splitting an identifier is to determine at which adjacent positions to insert splits. Thus, we can draw on the experience of the advancing thought to solve Chinese word segmentation[18]–**sequence labeling**. Similarly, identifier splitting can be subtly mapped to a sequence labeling task:

> For a given identifier id with characters sequence $(c_1, c_2, ..., c_n)$, we label each c_i with a corresponding tag. The set of candidate tags contains only B, M, E, S, and N.

Assuming the unified max-length of the accepted identifier is **T**, the interpretation of each tag is as follows: **B**, denoting the beginning of a term; **E**, denoting the end of a term; **M**, denoting the position between the beginning and end of a term; **S** denoting a term with only one character; and **N**, denoting the spare positions of an identifier if its length is less than *T*.

After tagging all c_i of id, we subsequently insert splits between any adjoining c_n and c_m with successive tags **EB**, **ES**, **SB**, or **SS**. For instance, we input identifier *nthreadpool* and obtain the output *SBMMMMEBMME*. Based on the rules summarized above, two splits are inserted between *n* and *t*, and *d* and *p*, respectively. Consequently, we obtain the splitting result *n-thread-pool*.

C. CNN-BiLSTM-CRF Model

According to the above discussions, the oracles of identifier splitting can be transferred into the form of sequence labeling accordingly. On this basis, we are able to train a **deep model** for labeling newly input identifiers, and then infer correct splittings of them. The considered deep model is actually our proposed CNN-BiLSTM-CRF model.

We vary the basic **BiLSTM-CRF**[19] network by adopting an additional CNN layer with the aim to extract finer-grained morphology features inside identifiers.

The architecture of CNN-BiLSTM-CRF is illustrated in Fig. 1, showing a slice of the entire network. The left side of Fig. 1 shows the general framework of BiLSTM-CRF. It differs from the traditional framework because the input layer of BiLSTM-CRF takes the replaced *CNN-processed char representation* of the identifier as input.

In particular, C_t denotes the input character at time t encoded using one-hot encoding[3], and $Conv_t$ denotes the t^{th} vector after convolution on the original input layer. To ensure one-to-one correspondence, we alternatively add padding around the beginning and end of the original input char representation. After that, we concatenate the output of the convolution layer and the original char representation to generate the final *CNN-processed char representation*. This gives the interpretation of $Conc_t$, which denotes the t^{th} concatenated vector. It is also worth mentioning that we skip the *pooling layer* with the aim of preserving all information of the input identifier in our task.

[3]We perform character-level embedding on each input character with a **one-hot** vector (with shape $1 \times n$). This means there is only one component equal to 1 in this vector. For example, C_t can be $[0, 1, 0, ..., 0]$.

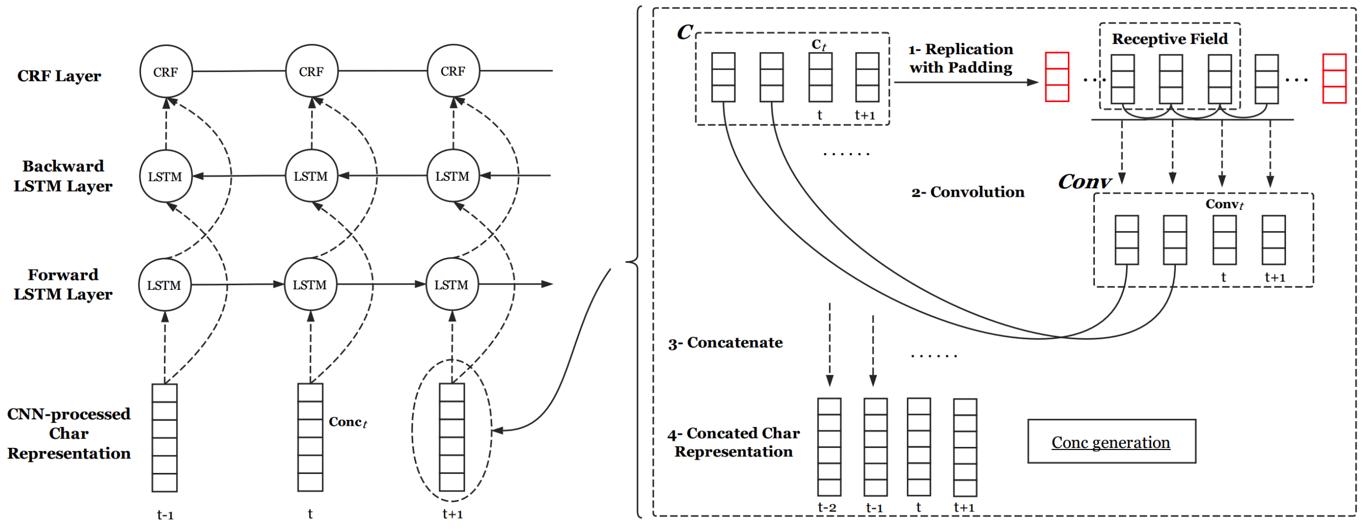

Fig. 1. Architecture of CNN-BiLSTM-CRF. Red rectangles indicate the padding for the CNN layer.

Suppose that K is the convolution core with shape $m \times n$ (n is equal to the length of dimensionality of the one-hot encoded vector). The convolution layer is calculated as follows:

$$Conv_i = f(\sum_i C'_{i+m} \circ K_m) \tag{1}$$

where symbol f denotes the activation function of *convolution* layer (normally the **Relu** function), the $\circ$ operation produces the *Hadamard product* of two vectors, and C'_i denotes the i^{th} vector of padded matrix C, defined as the original input layer in Fig.1.

Consequently, the concatenated representation of an input char is expressed by merging two corresponding vectors of $Conv_i$ and C_i, as shown in Equation 2.

$$Cont_{i_{(2\times n)}} = \left(Conv_{i_{(1\times n)}} \quad C_{i_{(1\times n)}}\right) \tag{2}$$

With regard to the remaining BiLSTM-CRF, CRF on the top of BiLSTM takes the final softmax layer of the BiLSTM layer as input. The softmax layer produces the probability distribution over labels (i.e., B, M, E, S, and N). The CRF layer calculates the **emission scores** and **transition scores** based on the probability. The emission score of a label can be directly expressed by reusing its output probability, while the transition score is a bit difficult to deduce. This requires us to maintain a 5×5 matrix (5 meaning the number of all possible labels), which records the transition score from one label to another label. In nature, the matrix is randomly initialized with small values. To elaborate, for a given output path p of an identifier $B \Rightarrow M \Rightarrow E \Rightarrow S$, we have

$$EmissionScore_p = X_{0,B} + X_{1,M} + X_{2,E} + X_{3,S} \tag{3}$$

$$TransitionScore_p = T_{B,M} + T_{M,E} + T_{E,S} \tag{4}$$

Therefore, for a path p, the final score of p is calculated as

$$Score_p = EmissionScore_p + TransitionScore_p \tag{5}$$

The final scores of other possible paths besides p are calculated in the same way. Among all paths, the path with the highest final score is actually thought to be the real path (i.e., correct path). After determining how to calculate the final score for each possible path, we are now able to define the CRF loss function:

$$LossFunction = -log(\frac{Score_{Realpath}}{\sum_{n=1}^{N} Score_n}) \tag{6}$$

During the training process, the parameter values (including the transition score matrix) of our BiLSTM-CRF model are updated repeatedly to continue increasing the percentage of the score of the real path. Because only bigram interactions between outputs are being modeled, the total scores for all possible paths can be computed using dynamic programming[20]. The log probability of the correct tag sequence is maximized by automatically reducing the loss of Equation 6. When the model is ready, we can infer the labels for the input identifier by using a Viterbi decoding algorithm based on the transition score matrix and the output of the BiLSTM layer.

III. Experimental Validation

In this section, we describe the series of experiments conducted on the four oracles mentioned previously. We chose 30 to be the unified length of the input layer because we found that identifiers with the length less than 30 occupy more than 96% of the samples on average for all datasets.

In addition, approximately 5,000 manually constructed identifiers[4] were collected as extra training materials, with which we could optionally supplement the training set partitioned from a given oracle. Specifically, these **fake-identifiers** were used in $\text{Experiment}_{\text{III-B}}$ and $\text{Experiment}_{\text{III-C}}$ to further enhance the performance of trained models.

The hyperparameters in use and details of the proposed CNN-BiLSTM-CRF structure are summarized in Table I.

[4]They were generated with the aid of *SCOWL* – *http://wordlist.aspell.net/*

TABLE I
THE HYPERPARAMETERS & STRUCTURAL DETAILS TO CONSTRUCT OUR PROPOSED CNN-BiLSTM-CRF MODELS

Component	**Hyperparameter**	**Value (Option)**
CNN layer	Number of layers	1
	Filter shape	$3 \times n^{\mathbf{1}}$
	Filter stride	1
	Activation function	Relu
BiLSTM layer	Number of layers	2
	Dropout probability	0.0
Fully connected layer	Number of layers[2]	1
	Activation function	None
General settings of training process	Initial learning rate	1.0
	Learning rate decay[3]	0.5
	Max gradient norm	5
	Max training epoch	13
	Batch size	25

[1] *As mentioned in Section II.C, the **n** is equal to the length of dimensionality of the one-hot encoded vector.*
[2] *The number of nodes of the hidden layer is set to 200.*
[3] *The learning rate will decay dynamically from one training epoch to next epoch based on an exponential decay model.*

A. Evaluation Measures

We use the measure **Accuracy** to evaluate the performance of our trained models. To calculate the accuracy, we need to measure whether every predicted split of an identifier is exactly equal to the corresponding correct split in the oracle (i.e., the percentage of correct splitting). Thus, it is simply calculated as follows:

$$Accuracy = \frac{Num(correctly\ split\ identifiers)}{Num(all\ identifiers)} \tag{7}$$

Several previous studies (e.g., [8], [13]) additionally used *Precision, Recall, and F-measure* to measure the local ability of evaluating techniques. However, we posit that these measures always follow the same trend with *accuracy* and behave extremely congruously, based on observations of our early research and previous experimental results. In addition, most other studies highly related to this topic used no other kinds of measures (e.g., *ROC-AUC* and *Cohen's kappa*[5]). Therefore, we pretermit considering other metrics in order to make our outcomes clear.

B. Investigation of the ability to split identifiers

This experiment was set up to validate the effectiveness of our proposed technique intuitively.

[5]There are two reasons for the unsuitablility of applying *Cohen's kappa* or *ROC-AUC* to this multiclass classification task: **1)** Our task additionally concerns the sequential relatedness of labels, and **2)** other baseline techniques used as benchmarks in this study infer identifier splitting on the *word level*, whereas our proposed technique is on a different *character level*.

TABLE II
ACCURACY OF IDENTIFIER SPLITTING COMPARED WITH OTHER STATE-OF-THE-ART APPROACHES

Splitting Approach	**Oracle**			
	Binkley	BT11	Jhotdraw	Lynx
CNN-BiLSTM-CRF	**0.817**	**0.936**	0.912	**0.876**
GenTest	0.701	0.723	0.795	0.489
INTT	0.774	0.820	0.844	0.625
LIDS	0.713	0.811	0.903	0.542
DTW	-	-	0.931	0.703
LINSEN	-	-	**0.949**	0.803

To assess the prediction performance of our trained models in a general and proper way, a **10-Time Hold-out Validation**[6] based on **Train/Validation/Test Set Splitting**[7] was used in this experiment. In other words, for every specific oracle, we iteratively trained and evaluated models 10 times, and the average of all results was subsequently used as the final estimation result on our models.

Several techniques (listed in Table II) were selected for benchmarking and were applied to split these oracles at the same time. Slightly different from our proposed technique, these compared techniques performed on all samples of the four oracles because no training processes were used in their cases. Specifically, we implemented LIDS[13], GenTest[7], and INTT[17] because they are provided with publicly available tools. For those that do not have publicly available tools, i.e., DTW and LINSEN, we tried obtaining their experimental results from study[15].

The experimental results are listed in Table II, with the best accuracies among all compared approaches in bold. The results show the following: On the Binkley dataset, CNN-BiLSTM-CRF achieves the highest accuracy (81.7%). On the Jhotdraw dataset, our proposed technique is suboptimal (91.2%) among all techniques. LINSEN seems to perform best on Jhotdraw since it has the highest accuracy (94.9%). However, on BT11 and Lynx, oracles with relatively more samples for training, the performance of our proposed technique is vastly superior to that of all other techniques. In particular, in terms of accuracy, CNN-BiLSTM-CRF outperforms the second best technique by more than 7%–11%.

We conducted a hypothesis test (**Pearson chi-square test** and **odds-ratio**) to investigate whether the difference in performance between CNN-BiLSTM-CRF and other state-of-the art techniques varies significantly. To simplify our results and

[6]One round of the *k*-time hold-out validation (a.k.a. *Monte Carlo Cross Validation*[21]) involves randomly partitioning samples into disjoint subsets, of which at least one is used as training set and at least one is used as test set. Then, the model fit by the training set is evaluated with the test set. This process is independently repeated *k* times, where the partitions of samples are accomplished in the same random manner for each run.

[7]In this *three-way split*, the original dataset is randomly partitioned into three parts at each time: 70% for the training set, 15% for the validation set to help avoid overfitting during the training process, and the remaining 15% to evaluate the performance of the model.

TABLE III
SIGNIFICANCE TEST FOR DIFFERENCE IN ACCURACY BETWEEN CNN-BILSTM-CRF AND OTHER STATE-OF-THE-ART APPROACHES

Oracle	**Compared Approach**	**Chi-square Test**	**Odds-ratio**
Binkley	INTT	p=0.03750(*Significant*)	1.35(Better)
	LIDS	p<0.00001(*Significant*)	1.80(Better)
BT11	INTT	p<0.00001(Significant)	3.21(Better)
	LIDS	p<0.00001(*Significant*)	3.41(Better)
Jhotdraw	DTW	p=0.4951	0.80(Worse)
	LINSEN	p=0.1069	0.59(Worse)
Lynx	DTW	p<0.00001(*Significant*)	2.95(Better)
	LINSEN	p<0.00001(*Significant*)	1.71(Better)

*[1] *The difference between two approaches is significant if the adj. p-value statistic is less than the significance level (0.05).*
*[2] *Odds-ratio greater than one means the treatment approach (CNN-BiLSTM-CRF) performs better than the compared control approach.*

make things clearer, we chose only the top two well performing techniques except CNN-BiLSTM-CRF for comparison. The statistical results of the hypothesis test are listed in Table III, from which we learn the following:

- CNN-BiLSTM-CRF performs significantly more accurately than other techniques on the Binkley, BT11, and Lynx datasets.
- Even though CNN-BiLSTM-CRF achieves lower accuracy than LINSEN on Jhotdraw, there is no significant difference between them with the p-value of a chi-square test of 0.1069, which does not exceed the critical value of 0.05. This means our technique, for Jhotdraw, performs at least in line with LINSEN (i.e., the technique with the best performed accuracy on Jhotdraw).

C. Investigation of external generality

This part of experiments was set up to explore the extensive feasibility of applying the CNN-BiLSTM-CRF model to split identifiers outside of the original oracle with which it was trained. Hence, to realize this aim, we trained models on four training sets, namely, *Binkley*, *BT11*, *Jhotdraw*, and *Lynx*. Then, we evaluated the accuracies of trained models by other evaluation sets.

The experimental results are shown in Table IV. The top header of Table IV lists the training sets with which we trained the models. The sidebar lists all evaluation sets. Hence, each cell inside the table refers to the average accuracy we obtained from the models that were trained and estimated with a specific pair of training and evaluation sets. It illustrates that the performance of models on other oracles which they were not trained with, by and large, approaches to the ideal performance reported in Table II (despite declines at different degrees). We even see a slight improvement in Jhotdraw from the model trained with BT11. Not surprisingly, the models trained with *Jhotdraw* perform worst since Jhotdraw contains a limited number of samples (only 974).

TABLE IV
EXTERNAL PERFORMANCE (i.e., ACCURACY OF SPLITTING) OF CNN-BILSTM-CRF MODELS

$\mathcal{S}_{\text{Training}}$ / $\mathcal{S}_{\text{Evaluation}}$	Binkley	BT11	Jhotdraw	Lynx
Binkley	-	0.760	0.655	0.703
BT11	0.833	-	0.796	0.814
Jhotdraw	0.896	0.936	-	0.883
Lynx	0.782	0.844	0.709	-
Total Avg.	0.829	0.825	0.773	0.805

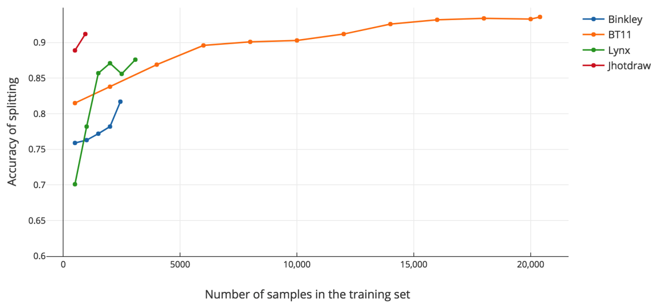

Fig. 2. Effect of training set size on model performance.

D. Investigation of effect of training set size

By picking and training samples with sizes from small to large in every fixed intervals from all datasets, we built CNN-BiLSTM-CRF models and then evaluated them by recording the related data of the accuracies. Similarly, for each size, we trained 10 models and acquired their average results. The results are shown in Fig. 2, which indicates that models trained on all different datasets follow the same trend. This reveals that as the size of training set increases, the performance of the model improves.

On the largest dataset BT11, however, it is noteworthy that the increment gradually slows down when the size of the training sets exceeds 6,000. It seems that 6,000 could be an appropriate size of training sets if we want to strike a **compromise** between the model performance and training cost under this circumstance.

E. Investigation of training cost

We determined four categories of training set size, namely *2000*, *8000*, *15000*, and *27000*, to survey whether our proposed technique is time-consuming. Specifically, we conducted 10 groups of experiments on four randomly constructed oracles which are corresponding to the above-mentioned four sizes. For each group, we separately trained four models on these oracles, and respectively recorded their training times (graphically provided in Fig. 3). Experiments from group *No. 1* to *No. 10* were carried out independently, but the plotting line of each corresponding size revealed strong consistency and stability on the training time.

To deeply inspect whether the computational time would be influenced by different constructions of samples, we randomly

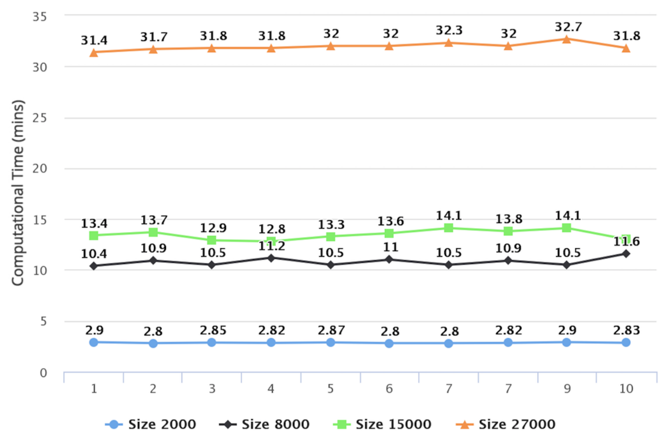

Fig. 3. Computational time of training models on training sets of different sizes. The *No.* on the x-axis represents that it is the i^{th} group of experiment. The value on the y-axis represents computational time of training a specific model once.

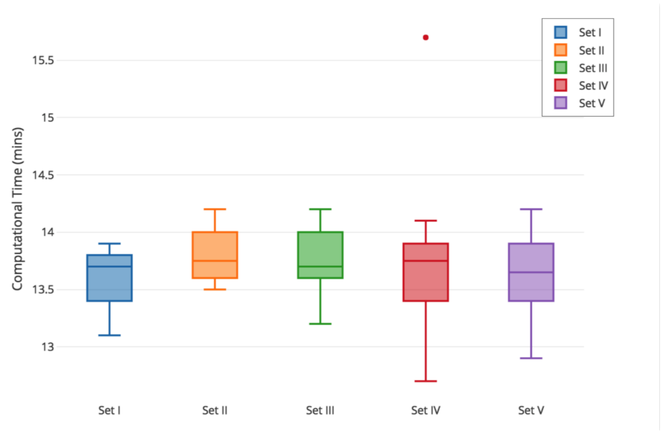

Fig. 4. Training cost of five sets of data with 15,000 samples.

constructed four extra training sets, all with a size of 15,000, to perform similar experiments. We used the *box plot* to measure the dispersion of the computational time of each set. The results are shown in Fig. 4, of which each horizontal line in a box refers to the average training time. We found out that models on all sets of samples finished their training with a nearly identical median of training time (**13.7 min**).

From all the illustrated results, we concluded that

There is not a significant amount of time for training (It takes about, on average, 10 minutes to train a model with 8,000 samples and 31 minutes with 27,000 samples.). In addition, the training time is shown to be sufficiently stable.

These data were produced and collected from *a PC with 1* NVIDIA P100, Octa-core 2.5-GHz CPU and 16 GB of RAM*.

IV. Discussion

A. Deeper insight into CNN-BiLSTM-CRF approach

We determined that the CNN-BiLSTM-CRF model is suitable for handling identifier splitting after pondering the relationship between this adopted model and domain-specific data (i.e., identifiers).

Through utilizing the characteristics of a CNN to perform character-grained feature extraction, the model is capable of learning the word morphology transformations. This provides an expansion ability for our technique to effectively handle multiple forms of a term. This will become more significant if models are trained on large datasets. Furthermore, the BiLSTM network enables the model to automatically learn contextual information, i.e., whether to insert a split in the current position according to characters in front and behind. Third, the CRF layer helps to eliminate those candidates splitting with few probabilities (e.g., tag M should never be followed by B or S). By integrating all of the needed abilities, our proposed technique certainly performs well.

We noticed from the above results that when our proposed technique was applied to oracles with a sufficient number of samples, the performance was significantly better than oracles with fewer samples. This is because neural networks always perform more poorly on small datasets. A larger training set provides richer information for a neural network to learn and to avoid overfitting. (Table II and Fig. 2 both strongly confirm this point.)

B. Possibility for improvement

After observing 200 incorrectly split identifiers inferred by our trained models, we figured out that a large proportion of them (93/200) were caused by a lack of training materials. This caused our models to fail to recognize terms inside these identifiers (e.g., *NXScanf*, where *Scanf* is the name of a C library function).

Given the extensibility of our technique, we are able to address this problem easily by mining more complete identifiers or constructing more customized *fake-identifiers*. These samples are then appended to the oracle before training a model. In addition, we believe that some unused deep learning or tuning skills provide the possibility of improving the performance from the model itself.

C. Is CNN-BiLSTM-CRF really feasible?

We deduce the answer of this question to be yes, based on all of the aforementioned outcomes. First, this proposed approach is superior if the training samples are customized and sufficient enough (according to the results in part B of Section III). Second, the model is of great extensibility. The results in Table IV indicate that most trained models achieve a total average accuracy of 80% when splitting those identifiers which they never met. This convinced us that it is entirely possible to construct a model with sufficient generality. Although we failed to obtain good performance on Jhotdraw, to some extent this is insignificant because we will certainly ensure that our models are trained on oracles with enough samples in practice. Third, training the model is feasible with regard to time. Only a little time is needed to construct an available CNN-BiLSTM-CRF model, and this is within the acceptable range. (Samples of all oracles are within 21,000 in our study, but all of them led to good performance). The cost of inferring a newly received identifier could even be omitted.

D. Threats to validity

The validity of our study has three major threats.

The threat to **content validity** is that using only four oracles as in our study seems to be insufficient for the hypothesis tests to be reliable. Although they contain identifiers that originated from several kinds of programming languages (i.e., C, C++, and Java), they still cannot represent all kinds of languages. Moreover, some of these oracles may not contain enough multiplex samples because of their small size.

Threats to **internal validity** include the hyperparameters of constructing the CNN-BiLSTM-CRF network and the randomness of the training process of a neural network. We selected the hyperparameters (e.g., the activation function and batch size) based on practical experience from the academic area.

The threat to **instrumental validity** is that we merely observed accuracy data on the Jhotdraw and Lynx datasets of LINSEN and DTW from a previous study. However, we still made a strong effort to compare our proposed technique with other state-of-the-art techniques on two other datasets.

We realized our approach with **Tensorflow 1.4**, a mature deep learning framework maintained by Google. The ready-made package to split identifiers, implemented code, oracles, and indications to reproduce our experimental results can be found in our Github repository[8].

V. Conclusion

We described a completely new approach, CNN-BiLSTM-CRF, to perform identifier splitting. In the premise of properly setting up the model, this approach is superior to other state-of-the-art techniques. We further investigated the feasibility of the proposed technique. The experimental results jointly demonstrated that the technique is practical and efficient with regard to training cost and the demand on the size of the training sets. The results also showed that this approach exhibits good generalization performance on various datasets, including splitting identifiers outside from trained-with oracles. Ultimately, splitting identifiers via CNN-BiLSTM-CRF is proven to be helpful in practice, in combination with all the qualitative and quantitative analyses in this study.

Detailed heuristics and processes of constructing CNN-BiLSTM-CRF models will be introduced in the next phase of our research. In the future, we will also improve the feasibility and generality of this approach by integrating *transfer learning* techniques (e.g., investigation of cross-programming-language identifier splitting).

Acknowledgment

Sincere appreciation to Li Sun, Juan Qiu, Robbie Xie, and Kanglin Yin, who provided us with valuable comments and tremendous encouragement. We are also grateful toward S. Butler, D. Binkley, and Madani et al. because it would not have been possible to complete this work without their previous substantial efforts on the topic of identifier splitting and their publicly available oracles.

[8]*https://github.com/jaki2012/IdentifierSplitting-SEKE2018*

References

[1] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, pp. 334–345, IEEE, 2015.

[2] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 87–98, ACM, 2016.

[3] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for java methods," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2016.

[4] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*, pp. 279–290, ACM, 2014.

[5] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis," in *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, pp. 71–80, IEEE, 2009.

[6] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, no. 3, pp. 261–282, 2006.

[7] D. Lawrie, D. Binkley, and C. Morrell, "Normalizing source code vocabulary," in *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pp. 3–12, IEEE, 2010.

[8] E. Hill, D. Binkley, D. Lawrie, L. Pollock, and K. Vijay-Shanker, "An empirical study of identifier splitting techniques," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1754–1780, 2014.

[9] S. W. Thomas, A. E. Hassan, and D. Blostein, "Mining unstructured software repositories," in *Evolving Software Systems*, pp. 139–162, Springer, 2014.

[10] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec: An enhanced tag recommendation system for software information sites," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pp. 291–300, IEEE, 2014.

[11] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?," in *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*, pp. 11–20, IEEE, 2011.

[12] H. Feild, D. Binkley, and D. Lawrie, "An empirical comparison of techniques for extracting concept abbreviations from identifiers," in *Proceedings of IASTED International Conference on Software Engineering and Applications (SEA'06)*, 2006.

[13] N. R. Carvalho, J. J. Almeida, P. R. Henriques, and M. J. Varanda, "From source code identifiers to natural language terms," *Journal of Systems and Software*, vol. 100, pp. 117–128, 2015.

[14] N. Madani, L. Guerrouj, M. Di Penta, Y.-G. Gueheneuc, and G. Antoniol, "Recognizing words from source code identifiers using speech recognition techniques," in *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pp. 68–77, IEEE, 2010.

[15] A. Corazza, S. Di Martino, and V. Maggio, "Linsen: An efficient approach to split identifiers and expand abbreviations," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pp. 233–242, IEEE, 2012.

[16] D. Binkley, D. Lawrie, L. Pollock, E. Hill, and K. Vijay-Shanker, "A dataset for evaluating identifier splitters," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 401–404, IEEE Press, 2013.

[17] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Improving the tokenisation of identifier names," *ECOOP 2011–Object-Oriented Programming*, pp. 130–154, 2011.

[18] N. Xue, "Chinese word segmentation as character tagging," *International Journal of Computational Linguistics & Chinese Language Processing, Volume 8, Number 1, February 2003: Special Issue on Word Formation and Chinese Language Processing*, vol. 8, no. 1, pp. 29–48, 2003.

[19] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[20] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *arXiv preprint arXiv:1603.01360*, 2016.

[21] Q.-S. Xu and Y.-Z. Liang, "Monte carlo cross validation," *Chemometrics and Intelligent Laboratory Systems*, vol. 56, no. 1, pp. 1–11, 2001.