Empirical Software Engineering
https://doi.org/10.1007/s10664-019-09702-z

Domain-specific machine translation with recurrent neural network for software localization

Xu Wang[1] · **Chunyang Chen**[2] · **Zhenchang Xing**[1]

Published online: 30 April 2019

Abstract
Software localization is the process of adapting a software product to the linguistic, cultural and technical requirements of a target market. It allows software companies to access foreign markets that would be otherwise difficult to penetrate. Many studies have been carried out to locate need-to-translate strings in software and adapt UI layout after text translation in the new language. However, no work has been done on the most important and time-consuming step of software localization process, i.e., the translation of software text. Due to some unique characteristics of software text, for example, application-specific meanings, context-sensitive translation, domain-specific rare words, general machine translation tools such as Google Translate cannot properly address linguistic and technical nuance in translating software text for software localization. In this paper, we propose a neural-network based translation model specifically designed and trained for mobile application text translation. We collect large-scale human-translated bilingual sentence pairs inside different Android applications, which are crawled from Google Play store. We customize the original RNN encoder-decoder neural machine translation model by adding categorical information addressing the domain-specific rare word problem which is common phenomenon in software text. We evaluate our approach in translating the text of testing Android applications by both BLEU score and exact match rate. The results show that our method outperforms the general machine translation tool, Google Translate, and generates more acceptable translation for software localization with less needs for human revision. Our approach is language independent, and we show the generality of our approach between English and the other five official languages used in United Nation (UN).

Keywords Software localization · Neural machine translation · Mobile apps

1 Introduction

Software localization is the process of adapting a software application to a particular language, culture, and desired "look-and-feel" of a target market. It is a fundamental part of

Communicated by: David Lo, Meiyappan Nagappan, Fabio Palomba and Sebastian Panichella

✉ Chunyang Chen
chunyang.chen@monash.edu

Extended author information available on the last page of the article.

development process for an application to reach global markets and customers. Software localization process includes many steps, including extraction of need-to-translate text from graphics, user interface (UI) elements, scripts or other media, creation and maintenance of terminology glossaries, translation of text to the target language, and adjustment and testing the text alignment in the UI or other functional elements.

Existing work on software localization focuses mainly on the pre-translation steps, i.e., extraction of need-to-translate text (Rich 2011; Wang et al. 2010; Xia et al. 2013). Some recent work (Muntés Mulero et al. 2012; Alameer et al. 2016) also look into post-translation UI and functionality issues, e.g., the problems with the visuals or the layout of the application due to the string length changes after translations into another language, and proposes automatic testing approach to identify such issues. However, the translation step, the core step in software localization has been neglected. The translation step is labor-intensive and often requires a significant amount of efforts from the development teams. According to the observation of Alshaikh et al. (2015), among 2,500 open source projects, about 32% of all localized resource files were committed more than 6 months after the default resource file was committed. One of the most important reasons is that projects often cannot find a proper translator to perform the translation.

A possible solution is to use machine translation tools developed in Natural Language Processing (NLP) community. Indeed, recent years have witnessed the mature and success of machine translation techniques, such as Google Translate[1] or Bing Translator.[2] Studies (O'Brien 1998; Muntés Mulero et al. 2012) show that the number of words processed per day by a human translator can be significantly increased, when an efficient machine translation engine is used and the human translator only needs to improve the machine translation results in the post-editing phase. Unfortunately, general-purpose machine translation engines are trained with general corpus and thus lack the knowledge about the terminology and writing style of a particular domain.

For the purpose of software localization, text of software applications has several characteristics that challenge the effectiveness of general-purpose machine translation engine. We summarize the challenges below with some English-Chinese translation examples.

- Many words have daily-life meanings, but as domain-specific terms they often do not need to be translated. For example, "Steam" can be translated as "蒸汽" (vapor), but as the software name "Steam" (an online game platform), it does not need to be translated.
- Different languages may have unique characteristics that require subtle attentions in translation. For example, there is no "tense" concept in Chinese words. Although it could be ignored in general translation without information loss, the software localization process may need to consider language unique characteristics to make users feel that software is originally designed in native language, for example, translating "Like Tweet" as "赞这篇推文" (give thumbs up for this tweet), instead of just "喜欢推特" (love tweet).
- Phrase-based statistical machine translation cannot properly deal with the word sequence in the translations when the sentence contains domain-specific terms (e.g., placeholder in the software text i.e., %s) or has a complicated structure. For example, it is difficult to translate sentences with complicated structure like having a subordinate

[1] https://translate.google.com
[2] https://www.bing.com/translator

clause, e.g., "*%1$s requires one or more Google Play services that are not currently available*".

Due to the above challenges, general-purpose machine translation engines often cannot address linguistic and technical nuance in translating text of software applications of a particular domain. Several techniques (Wu et al. 2008; Ren et al. 2009; Zhang et al. 2013) have been proposed to address the challenges in domain-specific[3] machine translation. For example, in the medical domain, Eck et al. (2004) use the Unified Medical Language System to improve the general machine translation model. Such domain-specific translation techniques assume the availability of a high-quality dictionary of domain-specific terms and translations (usually developed by domain experts). The dictionary is used to customize general-purpose machine translation engines with additional domain knowledge so that the translation results can be improved. Software has become ubiquitous, and today, it can be found in just about every device we use as consumers or in our professional life. Developing a domain-specific dictionary and customizing general-purpose machine translation engines for software of each application domain would not be cost-effective for software localization.

In this paper, we present a deep learning based domain-specific machine translator that is trained with a large corpus of text of software applications for different languages. Our training corpus consists of millions of aligned dual-language sentences pairs (e.g., English-Chinese) extracted from language packs of Android applications from Google Play. The translator can be adopted for translating text in mobile apps for app localization.

Our approach is of a two-stage model. Driven by our collected large-scale parallel corpus, we first build a dictionary about frequent translation pairs for direct mapping. We then train a Recurrent Neural Network (RNN) encoder-decoder model (Cho et al. 2014), the state-of-art deep learning architecture for machine translation, with the help of attention mechanism.

To address the domain-specific problem in translation, we incorporate category information in the training process by treating the category as a special input token of the training sentence pairs. To overcome the problem with rare non-English domain-specific words, we treat them as out-of-vocabulary words in the machine translation system. We customize the original machine translation system by the output of a word alignment mechanism. This word alignment mechanism preserves out-of-vocabulary words in the translation, but their positions in the translation are still learned by the RNN model.

We carry out the evaluation in terms of BLEU score (Papineni et al. 2002) (a widely used metric in machine translation) and exact match rate of the translation results by our method. Our approach achieves 49.51% BLEU score which outperforms the general machine translation, Google Translate (40.42%) in the testing dataset of Android application text, and machine translation model without incorporating the domain-specific information. We not only demonstrate the effectiveness of our translation results between English and Chinese, but also test its generality of our model in translating English into the other 4 official languages in UN. The results show that our method work well for software localization in different language pairs. We also demonstrate the usefulness of our model by conducting pilot study with human translators involved. The results show that human translators provided with translations achieved better performance in terms of both satisfactoriness and translation time.

The main contributions of our work are summarized as followings:

[3]Note that "domain-specific" in this work refer to the domain of the software engineering, instead of app category.

– To our best knowledge, this is the first work to develop a domain-specific machine translation model for software localization in software engineering domain. We propose a two-stage approach with not only a dictionary built by association rule mining, also a neural machine translation model incorporating domain knowledge.
– We are the first to collect a big dataset which is sufficient for effectively training the domain-specific machine translation model.
– The evaluation of our approach demonstrates the accuracy of our translation results which outperforms the general machine translation engine, Google Translate.

2 Background

Before explaining our method for machine translation, we first introduce the basic concepts of the two key techniques, i.e., word embeddings and recurrent neural network, that our approach relies on.

2.1 Word Embeddings

Word embeddings are dense low-dimensional vector representations of words that are build on the assumption that words with similar meanings tend to be present in similar context. Studies (Turian et al. 2010; Mikolov et al. 2013) show that word embeddings are able to capture rich semantic and syntactic properties (e.g., the morphological forms of one word such as quick, quickly) of words, compared with one-hot word representation (Salton et al. 1975). Many works (Chen and Xing 2016b; Chen et al. 2017b, 2018c; Huang et al. 2018) have demonstrated that word embedding works well in software engineering domain.

Continuous skip-gram model (Mikolov et al. 2013) is a recently proposed efficient algorithm for learning word representations using a neural network model. Figure 1 shows the general structure to train a continuous skip-gram model. The goal of the continuous skip-gram model is to learn the word embeddings of a center word (i.e., w_i) that is good at predicting the surrounding words in a context window of $2t + 1$ words ($t = 2$ in this exam-

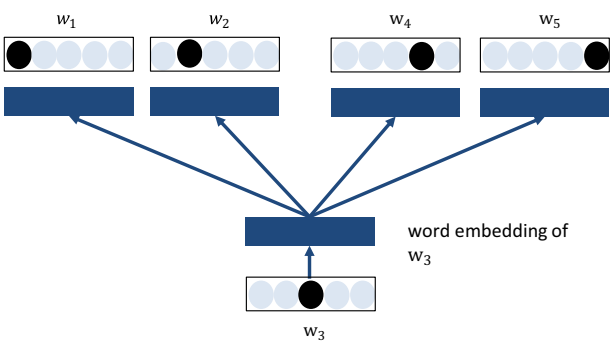


Fig. 1 Continuous skip-gram model

ple). More specifically, the objective function of the skip-gram model is to maximize the sum of log probabilities of the surrounding context words conditioned on the center word:

$$\sum_{i=1}^{n} \sum_{-t \le j \le t, j \ne 0} \log p\left(w_{i+j} | w_i\right) \tag{1}$$

where w_i denotes the center word in a context window of length $2t + 1$ and w_{i+j} denotes the context word surrounding w_i within the context window. n denotes the length of the word sequence. The $\log p\left(w_{i+j} | w_i\right)$ is the conditional probability defined using the *softmax function*:

$$p\left(w_{i+j} | w_i\right) = \frac{\exp(v'^{\mathrm{T}}_{w_{i+j}} v_{w_i})}{\sum_{w \in W} \exp(v'^{\mathrm{T}}_{w} v_{w_i})} \tag{2}$$

where v_w and v'_w are respectively the input and output vectors of a word w in the underlying a neural network, and W is the vocabulary of all words. Intuitively, $p\left(w_{i+j} | w_i\right)$ estimates the normalized probability of a word w_{i+j} appearing in the context of a center word w_i over all words in the vocabulary. This probability can be efficiently estimated by the negative sampling method (Mikolov and Dean 2013).

Given word sequences, the model maps words onto a low-dimensional, real-valued vector space. Word vectors are essentially feature extractors that encode semantic and syntactic features of words in their dimensions. In this vector space, semantically-similar words are also likewise close in term of their vectors.

2.2 Recurrent Neural Network

Recurrent Neural Network (RNN) is a class of neural networks where connections between units form directed cycles. Due to its nature, it is especially useful for tasks involving sequential inputs such as code completion (White et al. 2015), speech recognition (Graves et al. 2013), text classification (Chen et al. 2018a), and API document encoding (Chen et al. 2019). Compared with traditional n-gram language model (Mikolov et al. 2010), a neural language model based on RNN can predict a next word by predecessing words with long distances rather than a fixed number.

The network of RNN includes three layers, that is, an input layer which maps each word to a vector such as word embedding or one-hot word index, a recurrent hidden layer which recurrently computes and updates a hidden state after reading each word, and an output layer which estimates the probabilities of the following word given the current hidden state. Figure 2 shows the unfolding in time of the computation in RNN's forward computation. At

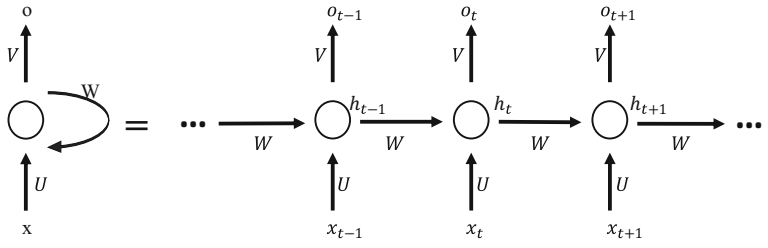


Fig. 2 Recurrent neural network

time step t, it estimates the probability of the following word $p(w_{t+1}|w_1, ..., w_t)$ by three steps. First, the current word w_t is mapped to a vector x_t by the input layer.

$$x_t = input(w_t) \tag{3}$$

Then, it generates the hidden state h_t in hidden layer according to the previous hidden state h_{t-1} and the current input x_t

$$h_t = h_{t-1} * W + x_t * U \tag{4}$$

where W, U are parameters inside the neural network. Finally, the $Pr(w_{t+1}|w_1, ..., w_t)$ is predicted according to the current hidden state h_t:

$$Pr(w_{t+1}|w_1, ..., w_t) = g(h_t) \tag{5}$$

During training, the parameters are learned by backpropagation (Werbos 1990) with gradient descent to minimize the error rate.

3 Data Collection

Unlike general machine translation which can easily utilize translation corpus (e.g., United Nation's documents for different languages) for model training, the biggest challenge for domain-specific machine translation is the lack of domain-specific translation corpus (Plamada and Volk 2013) that includes sufficient domain-specific bilingual sentence pairs. It is also labor-intensive task to extract language packs of different softwares, which involves a lot of human efforts such as crawling software packages, reverse engineering, sentence alignment, etc. In addition, as the language pack of each software normally contains only hundreds of sentences or phrases, it will require a large number of software applications to accumulate enough corpus for training a neural machine translation model.

According to our study, mobile applications are always of multi-language packs which users can select as their preferred display language. Therefore, we crawl Android apps (i.e., APK file) randomly from Google Play (2018a). Within the crawling process, we take the top 100 apps as the seed, and then crawl all their related apps recommended by Google. We iterate this process for one month from Jun 15, 2017 to Jul 15, 2017. Finally, we totally collect 108,339 apps from Google, and these apps belong to 25 categories[4] such as education, entertainment, music & audio, etc in Fig. 3.

As each APK is an executable file, we use Apktool (2018) to decompile the packages and obtain the resource folder which contain all external files . Note that all strings displayed in the Android app is stored in one file called "strings.xml" (containing English), and all translations for the file are stored in separate files which are in folders ended with language abbreviation[5] such as *values-fr*, *values-ja*, and so on. There are 6 official languages used in United Nations (2018b) including Arabic, Chinese, English, French, Spanish and Russian. As English is always the first language adopted in apps in Google Play, the software localization in this work is targeting at translating English into the other 5 UN languages.

[4] Although Google Play distinguishes detailed game category such as cards, racing, puzzle, we take them as one game category.

[5] https://developer.android.com/guide/topics/resources/localization

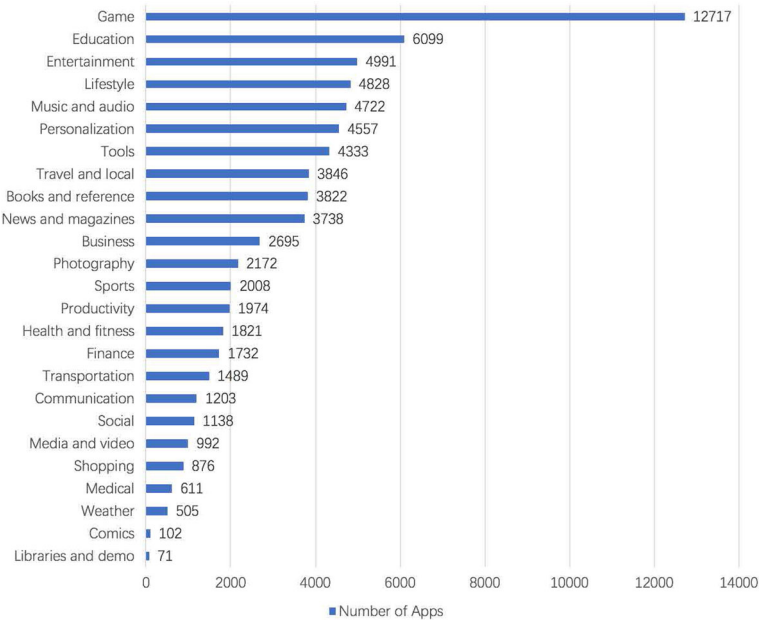


Fig. 3 The App distribution in different categories

Therefore, we extract string files of language pairs, and we can see that each piece of text is attached with a unique ID as seen in Fig. 4. Text in different language is aligned by these unique IDs within the app, and we build the parallel corpus in this way.

Table 1 shows that how many parallel sentence pairs that we collect between the target language and English. To our best knowledge, this is the largest multi-lingual parallel app-specific corpus in Software Engineering domain, and it can enable more future works in this direction. Note that not all apps contain these 6-language packs, so more than 79K apps contain Chinese-English and Russian-English pairs, while only 77K apps contain other language pairs. We then remove the duplicate pairs (i.e., pairs in which both source language and target language are the same.) and collect millions of sentence pairs for each language pairs. The sentence length varies across different apps and different languages, we calculate the distribution of sentence length in each language in Fig. 5. Most sentences are of short size. For example, 68.1% app sentences in English have no more than 4 words.

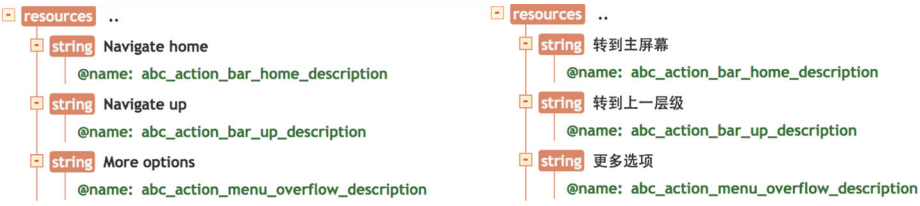


Fig. 4 Process of aligning XML files and forming training pairs

Table 1 Number of sentences and application for each language

Language	#Apps	#Sentences	#Sentences without duplications
Arabic	77,150	6,181,892	1,123,925
Chinese	79,145	6,419,324	780,044
French	77,283	8,297,821	1,508,657
Russian	79,451	7,691,341	1,199,439
Spanish	77,437	8,845,659	1,570,600

4 Software-Specific Translation

Considering the characteristics of mobile apps, we propose a two-stage approach for the app localization. First, given the input sentence of one language, if it frequently appears in our corpus, it is directly translated into the target language (Section 4.1). Otherwise, it will be sent to a neural translator which is build on millions of parallel sentences collected from apps (Section 4.2). Our neural translator consists of an RNN encoder and decoder model with attention mechanism(Section 4.2.1) by incorporating app-specific category information(Section 4.2.2), and copying rare domain-specific terms(Section 4.2.3).

4.1 Dictionary-Based Translation

According to our observation, many apps share some similar text such as *log in*, *register*, *search*, as these functionalities are common across different apps. Instead of using neural network based translation, we adopt a light-weight dictionary to directly translate those commonly-used sentences. We first collect all translation pairs, source sentence (s), target sentence (t) in two parallel languages. Then we adopt the bigram association rules mining (Agrawal et al. 1993; Chen and Xing 2016a; Chen et al. 2016b) for building the

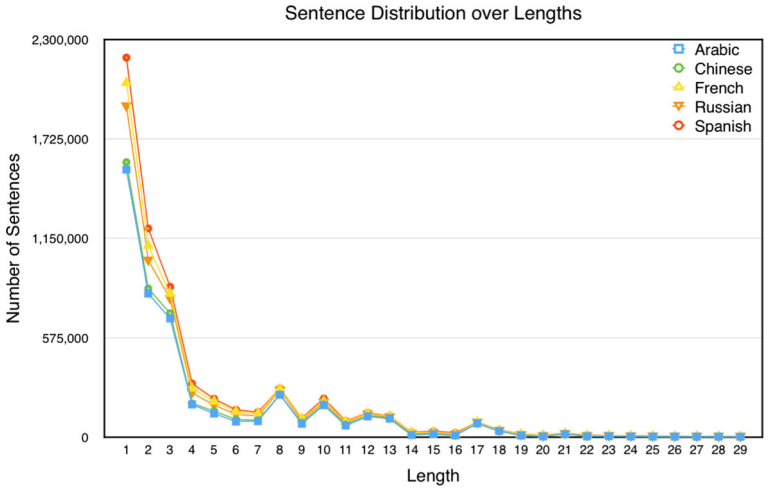


Fig. 5 The distribution of sentence length

software-specific translation dictionary (Table 2). There are two parameters in association rule mining:

$$support(s,t) = \frac{\#translation\ pairs\ containing\ (s\ and\ t)}{\#all\ translation\ pairs}$$

$$confidence(s \Rightarrow t) = \frac{\#translation\ pairs\ containing\ (s\ and\ t)}{\#translation\ pairs\ containing\ s}$$

The *support* value measures how frequently the translation pairs appear in all translation pairs. The *confidence* value measures the proportion of the translation pairs containing both s and t compared with all the translation pairs containing s. Note that traditionally, the rule mined from association rule mining always contains more than one items in each side e.g., {A,B, C $->$ D,E}. As we are trying to assist translation, each side can only contain one item, i.e., either source or target. To distinguish our approach from traditional ones, we call it bigram association rules mining.

We use the association rules mining rather than frequent itemset (Borgelt 2012) due to our observation that some terms may be translated into different words in the target app info. For example,"View" can be translated into "浏览"(read) in 875 apps, but translated into "查看" (check out) in 4516 apps. To determine which sentences can be directly translated, we define *confidence* value to show how united one sentence can be translated into the other language. The higher score represents that the sentence is more likely to have a common translation regardless of the category.

However, the higher score also leads to fewer sentences translated by the dictionary. To analyze the influence of different confidence values to the exact match rate and translated sentences, we carry out an experiment. The results can be seen in Fig. 6 that the exact match rate and proportion of affected sentences are negatively related. To achieve the balance between exact match score and translated sentences i.e., translate more sentences while preserving reasonably good performance, we set confidence value at 0.5 in our experiments. The dictionary built from that value can translate over 31.4% of the sentences and achieves 82.6% exact match rate.

Table 2 Most frequent sentences and phrase and corresponding translation in Chinese

Common english sentence	Chinese translation
Share	分享
Update	更新
Done	完成
More options	更多选项
Sign in	登录
Try again	重试
Navigate home	返回主界面
Open on phone	在手机中打开
Choose an app	选择应用
Get Google Play services	获取Google Play服务

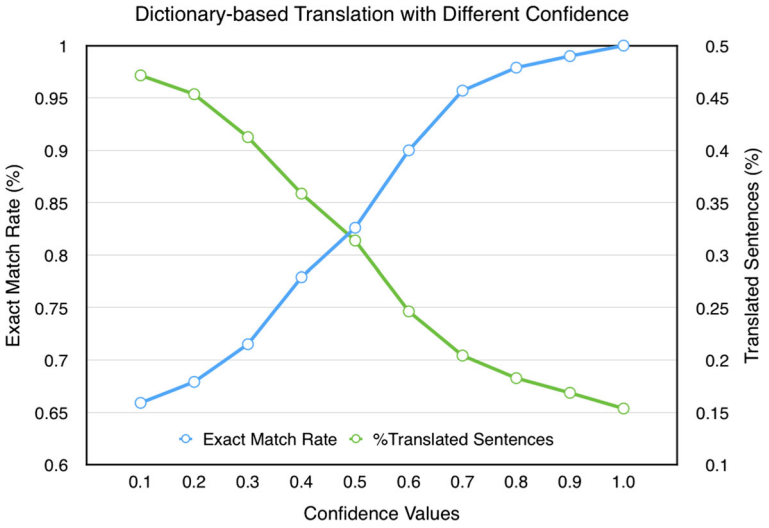


Fig. 6 Performance and influence of our dictionary with different confidence values

4.2 Neural Translation

4.2.1 RNN Encoder-Decoder Model

This model includes two RNN as its main components: one RNN to encode a variable-length sequence into a fixed-length vector representation, and the other RNN to decode the given fixed-length vector representation into a variable-length sequence. From a probabilistic perspective, this model is a general method to learn the conditional distribution over a variable-length sequence conditioned on yet another variable-length sequence, i.e., $p(y_1, ..., y_{T'}|x_1, ..., x_T)$. Note that the length of the input T and output T' may differ.

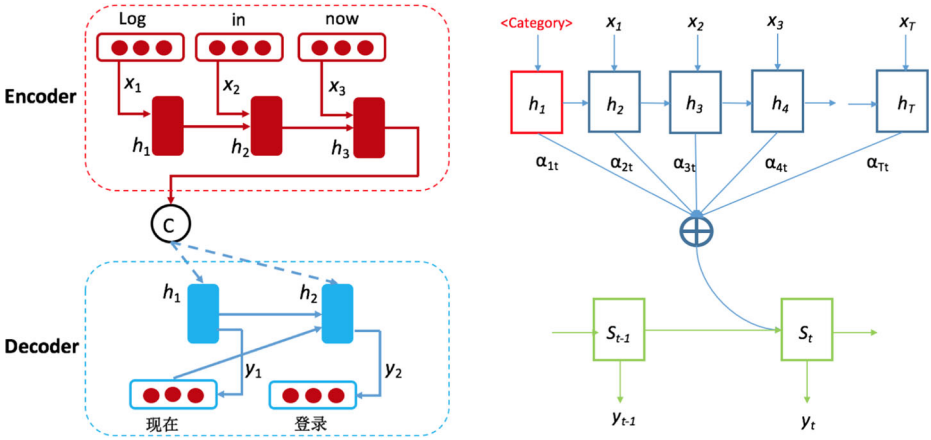
The architecture of this model for the purpose of software text translation can be seen in Fig. 7a. The encoder is an RNN that reads each word of an input sequence x sequentially. As it reads each word, the hidden state of the RNN changes according to (4). After reading the end of the the input (marked by an end-of-sequence symbol), the hidden state of the RNN is a summary $\mathbf{c}$ of the whole input sequence.

The decoder of the model is another RNN which is trained to generate the output sequence by predicting the next word y_t given the hidden state $h_{(t)}$. However, unlike the basic RNN architecture described in Section 2.2, both y_t and h_t are also conditioned on y_{t-1} and on the summary feature vector $\mathbf{c}$ of the input sequence. Hence, the hidden state of the decoder at time t is computed by,

$$h_t = f(h_{t-1}, y_{t-1}, \mathbf{c}) \tag{6}$$

and similarly, the conditional distribution of the next word is

$$Pr(y_t|w_1, ..., w_{t-1}, \mathbf{c}) = g(h_t, y_{t-1}, \mathbf{c}) \tag{7}$$



(a) The basic structure of the RNN encoder-decoder model

(b) RNN encoder-decoder with attention and categorical information

Fig. 7 Neural translation model

The two components of the RNN encoder-decoder are jointly trained to maximize the conditional log-likelihood

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} logp_{\theta}(y_n|x_n) \tag{8}$$

where θ is the set of the model parameters and each (x_n, y_n) is a pair of input and output sequence from the training corpus. The model parameters are estimated by backpropagation with a gradient-based algorithm.

4.2.2 Adding Attention and Category Information

In the original basic architecture of RNN Encoder-Decoder, the decoder is supposed to generate a translation solely based on the last hidden state from the encoders.

But such vector may not encode all the information, especially for long sentences, so the decoding process based on it will lead to worse results. According to our observation, different parts of input could have different importance to the words in the expected translation. For example, considering the input "*This device does not support this function*", and the target translation is "此设备不支持此网络" , the word "*support*" is more important than "*not*, *this*" to the target translation word "支持". Therefore, to avoid the limitation of the vanilla RNN Encoder-Decoder, we adopt the attention mechanism (Bahdanau et al. 2014) in this work.

With the attention mechanism, we no longer rely on merely the last hidden output to represent the entire input description. Instead, we allow the decoder to "attend" to different parts of the source sentence at each step of the output generation. Figure 7b illustrates the workflow of the attention mechanism. In the figure, y_t is our generated word by the decoder at the time step t, and the $x_1, x_2, \ldots x_T$ are our source sentence words. The attention model defines individual context vector c_i's for each target word y_i as a weighted sum of

all historical hidden states $h_1, \ldots, h_{T_x}$, not just the last state in the original RNN Encoder-Decoder. That is,

$$c_i = \sum_{t=1}^{T_x} \alpha_{it} h_t \tag{9}$$

Where the α_{it} is the weight that defines in how much of the input x_i should be considered for the output y_t. So, if the $a_{4,3}$ is a large number, this would mean the the decoder pays much attention to the third state in the source sentence while generating the fourth word of the target method name. The context vector c_i depends on a sequence of hidden states to which an encoders maps the input sentence. Each hidden state h_i contains information about the whole input sequence with a strong focus on the parts surrounding the $i-$th word of the input sequence.

A big advantage of attention is that it gives us the ability to interpret and visualize what the model is doing. For example, by visualizing the attention weight matrix a when a sentence is translated, we can understand how the model is translating from Fig. 8. The heat map shows the attention weights of each word in descriptions for the subtokens in method name.

In Fig. 8b, given the input "*<transportation > please contact your driver*", the word "driver" leads to "司机" in the translation, and our model predicts "司机" by seeing the surrounding words "contact", "your" , and the input category "transportation". However, in Fig. 8a, given the input "*<tools> select your wifi driver*", the word "driver" leads to the different translation "驱动程序" (launcher) by seeing the word "wifi" and category information "tools". These examples show the power of the attention mechanism which can spot the alignment between the input and the output to help direct the translation.

Apart from the attention mechanism, we also add some domain-specific external information to the RNN encoder-decoder model. As we know, the apps in Google Play are separated into different categories such as game, education, tool, music, etc. According to our observation, the same word in one language may be translated into different words in the other languages, as the word in the source language may have different meanings in different context. For example, retrieve is translated to "取单" (collect the food delivery)in a business application "Point of Sale", However, the same word is translated to "搜索" (search) in a travel application "Fly Scoot".

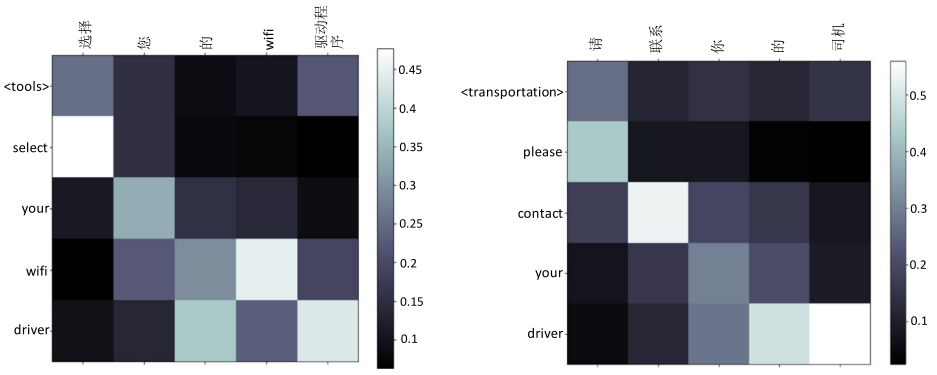


Fig. 8 Attention visualizations

To incorporate the external category information to the model, we first convert the 25 categories into vectors. We assign each category with a specific 25-dimension one-hot vector i.e., with only one position as 1 but other positions as 0 in the vector. To make the model easier to train, we append 0 for each category vector so that the size of category vectors is equal to that of other word embeddings. Then for each input sentence, apart from converting word vectors of the sentence, we also append the vector of its category where its app belong to. Therefore, the decoder will also consider the category information in the translation process. For instance, in Fig. 8a, given the category "tools", "driver" is translated to '驱动程序' (a program that controls devices). However, in Fig. 8b, "driver" is rather translated to "司机" (people who drives vehicles) in a transportation application.

4.2.3 Addressing Rare Domain-Specific Word Problem

Due to the computationally intensive nature of the RNN training, neural machine translation systems often limit the size of vocabularies, which is usually the top 30K–80K most frequent words in each language (Mikolov et al. 2011). That is, they are incapable of translating rare words which are out of the vocabulary. This is an especially severe challenge for translation of software application text, because there are many domain-specific terms which rarely appear and thus are out of frequent-word vocabulary, such as a placeholder (e.g., *%1$s,*), app name (e.g., *XimalayaFM*), web address (e.g., http://www.ekiga.org), email (e.g., *rich@kde.org*) in Table 3.

When dealing with such out-of-vocabulary words, the original RNN encoder-decoder model will mark them as **UNK** in the translation results. For example, the sentence "*Check out %1$s's Tweet:* https://twitter.com/%1" includes two rare words i.e., "*%1$s*" and "https://twitter.com/%1". Although the whole sentence can be well translated by the RNN encoder-decoder model except these two rare words, the unknown mark severely influence the reading of the translation for human users, and thus cannot satisfy the goal of software localization.

According to our observation, unlike the general text, many terms in mobile applications are not necessary to translate such as placeholders, web addresses and other examples above. They do not have the counterpart translations in another language and it would better to preserve them in the translation results. In this work, we customized the original RNN encoder-decoder model specifically for software-application-text machine translation by applying a copyable rare word model (Luong et al. 2014). Given a pair of English and Chinese sentences for training the model, we first tokenize them. Then for each word in the English sentence, we check whether it is in the vocabulary, and, if not, the word will

Table 3 Rare word examples

Rare word types	Examples
Placeholders and tags	%s, %1$s,
App names	xiami, Doreso, XimalayaFM
Web adress	facebook.com/device, twitter.com/RestainoAnthony
Email	support@voclab.com, user@example.com
Numbers and metrics	999+, GHz, Mbps
Special characters	●, ©
Dates	2018/03/19, MM/dd/yy

be marked as a rare word, i.e., UNK_{index} where $index$ is a unique index of a rare word in the training corpus. For instance, for sentence "*Sent from my %1$s using %2$s, powered by appyet.com*" in Fig. 9, "*%1$s*", "*%2$s*" and "*appyet.com*" are marked as UNK_1, UNK_2, UNK_3 after preprocessing the training sentences. The rare words in the corresponding translation in Chinese are also marked using the corresponding UNK_{index} i.e., the "*%1$s*", "*%2$s*" and "*appyet.com*" in the target sentence are also marked as UNK_1, UNK_2, UNK_3, but note the order differs. The preprocessed English sentence and Chinese sentence are regarded as a training pair to train the RNN encode-decoder model. Given an English sentence to be translated, the trained RNN encode-decoder model translates it into a Chinese sentence. Note that the order of UNKs may be different between the source sentence and the target sentence, while our model can learn to align the order from the big training data. The special UNK_{index} symbols in the translation are then mapped back to the original rare domain-specific words in a post-translation step.

Note that the relative positions of these rare words in the translation may differ from those in the English sentence. However, with enough training data, the RNN encoder-decoder model can learn the relative position relation between the source sentence and the translation result.

5 Evaluation

We evaluate the effectiveness of our translation method by measuring the quality of its translation of software text, against human reference translations. Specifically, our evaluation addresses the following questions:

- RQ1: How accurate is our translation method, compared with general machine translation and the original RNN-based machine translation?
- RQ2: How accurate is our translation method for translating different apps?
- RQ3: How generalized is our model to other language pairs?
- RO4: How useful is our model to human translators for software localization?

5.1 Evaluation Metric

5.1.1 BLEU Score

Let $\langle s_g, t_g \rangle$ be a pair of translations in the testing dataset. We say t_g is the ground-truth target translation for source input s_g. Let t_p be the predicted translation for the s_g using our domain-specific translator.

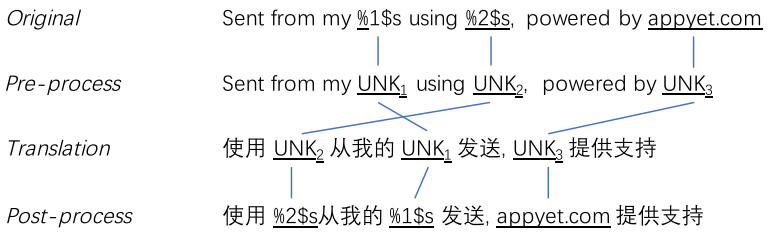


Fig. 9 The process to deal with rare domain-specific words

The first metric we use is *exact match rate*, i.e., the percentage of testing pairs whose t_g exactly match t_p. Exact match is a binary metric, i.e., 0 if any difference, otherwise 1. It cannot tell the extent to which a predicted translation differs from the ground-truth translation. For example, no matter one or 100 differences between the two translations, exact match will regard them as 0.

Therefore, we use BLEU (Bilingual Evaluation Understudy) (Papineni et al. 2002), an automatic evaluation measure widely used in machine translation studies. BLEU automatically calculates the similarity of machine-generated translations and human-created reference translations (i.e., ground truth). BLEU is defined as the product of "n-gram precision" and "brevity penalty" as:

$$BLEU = BP * \exp\left(\sum_{n=1}^{N} w_n log p_n\right) \tag{10}$$

where each p_n is the precision of the n-grams, i.e., the ratio of length n word subsequences generated by the machine translation system that are also present in the human reference translation. w_n is the weight of different length of n-gram summing to one. It is a common practice (Sutskever et al. 2014) to set N as 4 and $w_n = \frac{1}{N}$. However, some texts of software are very short sentences, especially text in UI components such as "file" or "save", which are shorter than 4 grams. Therefore, for long sentences, we still set N as 4 and $w_n = \frac{1}{N}$. For sentences whose length is smaller than 4, we set N as the minimum length of machine-generated translation and human reference translation and $w_n = \frac{1}{N}$.

BP is the brevity penalty which prevents the system from creating overly short hypotheses (that may have higher n-gram precision). BP is defined as

$$BP = \begin{cases} 1 & c > r \\ e^{(1-\frac{r}{c})} & c \leq r \end{cases} \tag{11}$$

where r is the length of human reference translation, and c is the length of machine-generated translation. BLEU gives a specific real value with range [0,1] and is usually expressed as a percentage. The higher the BLEU score, the similar the machine-generated translation is to the human reference. If the translation result completely equals to the human reference, the BLEU score becomes 100%. In the following paper, we will use the percentage form of the BLEU score.

5.1.2 Exact Match

One most important metric of evaluating a NMT system which is used for software localization is to what extent its translation can be directly used for localization. In this work, we use exact match rate to measure this ability of NMT systems. Exact match is defined as the proportion of translations which are exactly same as the ground truth:

$$ExactMatch = \frac{\#exactly\ translated\ sentences}{\#all\ testing\ sentences} \tag{12}$$

Note that although the higher exact match rate always leads to higher BLEU score, but these two metric do not rigidly positively correlate.

5.2 RQ1: Comparison with Existing Machine Translation Techniques

5.2.1 Evaluation Setup

In this experiment, we use 663,037 data of our translation corpus as the training data, 58,503 data as the validation data to tune the model parameters, and the rest 58,504 data for testing the translation results.[6] We have checked many software localization tools such as Transifex,[7] Crowdin[8] and smartling.[9] Most of them provide a platform to assist developers and translators collaboratively work on software localization. Many of them do not provide translation functionality but ask the translator to give the manual translation. Some of the platforms (e.g., Transifex) directly embed existing automatic translation tool like Google Translate inside. Therefore, we take Google Translate as the baseline for the comparison. We compare our domain-specific translation method with Google Translation in term of the BLEU score and exact match rate. We use Google Translate API[10] to obtain the translation results for the testing data. Our model is based on neural network, so we also compare with phrased-based machine translation model. In this work, we adopt the state-of-the-art phrase-based machine translator, Phrasal (Green et al. 2014) as the baseline. To be fair, we have developed a domain-specific tokenizer and fine-tune the parameters for Phrasal. In addition, we examine if our customized RNN encoder-decoder model can improve the BLEU score over the original RNN encoder-decoder model. Finally, we further explore the relationship between translation accuracy and sentence length to better understand our model. For brevity, we refer to the Google Translate as **GT**, the original RNN encoder-decoder as **2RNN**, and our customized model as **C-2RNN**.

5.2.2 Results

The BLEU score of the translation results by different translation methods can be seen in Table 4. The performance when applying original 2RNN without any customization is slightly worse than that of Google Translate. That is because Google Translate now also leverages neural-machine translation model as their backbone (Wu et al. 2016), and their model is rather large with tens of multi-layer neural network trained on tremendous corpus for long time which can only be afforded by Google. Therefore, it is very likely that Google Translate outperforms the naive 2RNN even if it is trained on domain-specific dataset. The Phrasal has better performance than that of Google Translate, and it has achieved 41.65% BLEU score and 0.44 extract match rate.

With adding attention, category, or dictionary, the performance of 2RNN has been boosted to some extent. For example, in term of exact match rate, the attention, category information, copy mechanism and dictionary can bring 30.7%, 27.9%, 29.0% ,48.4% increase to the original 2RNN model. The results show that our model which incorporates all customization steps significantly outperforms the other three baselines. Our model can achieve 49.51% BLEU score, which is 22.5%, 29.9%, 18.87% higher than that of Google

[6] We do not use cross validation for evaluation as the training process takes a long time on our PC.

[7] https://www.transifex.com/

[8] https://crowdin.com/

[9] https://www.smartling.com/

[10] This indeed limits the scale of our experiment because it is a paid service to use Google Translate API for large-scale translation (https://cloud.google.com/translate/v2/pricing).

Table 4 Translation accuracy

Approach	BLEU %	Exact match
GT	40.42	0.249
Phrasal	41.65	0.44
2RNN	38.12	0.283
2RNN with attention	42.55	0.37
2RNN with category	39.8	0.362
2RNN with copy mechanism	41.66	0.365
2RNN with dictionary	44.69	0.42
C-2RNN	49.51	0.471

Translate, the original RNN encoder-decoder model, and Phrasal respectively. In term of extract match rate, our model also obtain the best performance with 0.471, which is also 89.2%, 66.4% higher than the other two baselines.

For reference, an earlier study (Holzer et al. 2011) conducted in 2011 shows that Google Translate can achieve a BLEU score of about 49% in the general English-Chinese translation. However, even with several updates of Google Translate during the past 5 years, its BLEU score for software text translation is only 40.42%. The reason is that Google Translate is designed for common translation tasks and lacks support for translation under domain-specific scenarios. For instance, consider the sentence "be capable with any launchers" in Table 5. This sentence belongs to an application in "tools" category, and the word "launcher" has to be translated to "桌面" (desktop). However, as Google Translate do not incorporated the category information, this word is translated to "发射器" (ejector or projector), which is improper in the scenario of a desktop tool application.

Besides, the sentence "boost your memory now" should be translated to "现在加速您的内存". However, Google Translate translated "memory" to "记忆力" (human memory), while it actually refers to the RAM of a mobile device and should be translated as "内存". This confirms our assumption that general translation is not suitable for software localization tasks.

The results of our model also demonstrate that our model has the ability to tackle the domain-specific translation problems discussed in Section 1.

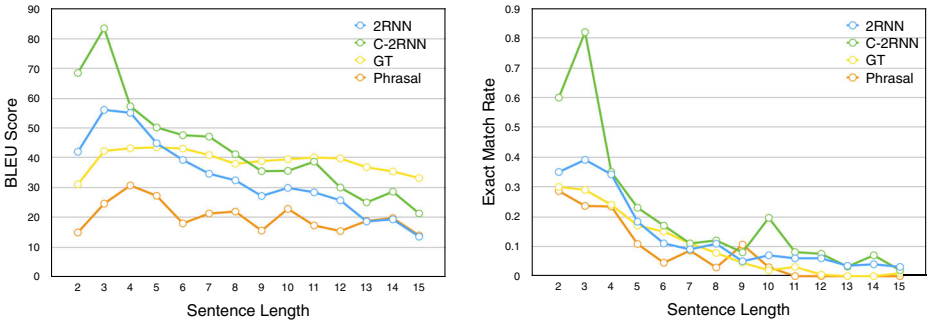
- According to Table 5, in sentence "start new Granny game", the word "Granny", which refers to the name of a game instead of actual grandmother, is preserved in the translation of our model. This indicates that our model can correctly handle the translation of domain-specific words.
- The sentence "keep device alive" in Table 5 is translated to "保持设备运行". If the word "alive" is translated to "活着" (living) such as Google Translate did, users may still able to understand this sentence but it sounds weird in Chinese since "活着" is often used to describe living creatures. However, though "运行" (running) has no direct relationship with word "alive", our model still picks the most nature and proper translation given the subject of the sentence is "device".
- Google Translate and other machine translation tools lacks support for translating domain-specific tokens such as placeholders and XML texts. For example, the translation of placeholders "%1$s" is "%1 $ s", which includes undesired spaces; Translation of "<heading> Reminder < / heading>" is "<heading> 提醒 < / heading>", where

Table 5 Translation examples: our model C-2RNN versus Google Translation

Source	Category	C-2RNN	GoogleTranslate
credit	Business	积分 (points)	学分 (academic credits)
generic wizard	Personalization	通用向导 (guide)	通用巫师 (sorcerer)
be capable with any launchers	Tools	与任何桌面兼容 (desktop)	与任何发射器兼容 (ejector)
report comment	Communication	举报评论 (inform offence)	报告评论 (neutral meaning of report)
max export size	Media and Video	最大输出尺寸 (output)	最大出口尺寸 (sell overseas)
are you sure to dismiss this group	Comminication	您确定关闭此群吗 (close, chat group)	你确定要解雇这个小组吗 (fire, team)
keep device alive	Lifestyle	保持设备运行 (running)	保持设备活着 (living)
boost your memory now	Tools	现在加速您的内存 (accelerate, RAM)	现在提升你的记忆力 (improve, human memory)
start new Granny game?	Games	是否开始新的 Granny 游戏(App name)?	开始新的奶奶游戏? (grandma)
web	Productivity	网页 (website)	卷筒纸 (paper volumn)
<heading> Reminder < /heading>	Tools	<heading>提醒 < /heading>	<标题>提醒</标题> (heading should be preserved)

"heading" in the angle brackets are supposed to be preserved in the translation since they are part of the XML grammar. Refining these translations from Google Translate requires extra time and labor. This problem is tackled by our model which has elaborately designed data processing pipeline. The translations of the above two example are the same as expected.

We then further explore those sentences that are not translated very well by our method. We find that the length of the sentence is the most crucial factor. As Fig. 10 shows, the longer



(a) BLEU Score (b) Exact Match

Fig. 10 The relationship between the BLEU score / exact match rate and the sentence length

the sentence, the lower the BLEU score for the three models. For short text, our model has much higher BLEU score and exact match rate than that of baselines. But considering the decreasing ratio, we can see that our customized RNN model decrease relative faster in term of the BLEU score, compared with Google Translate and original RNN model. It means that when dealing with the long sentences such as sentences with more than 15 words, the vector representation may not capture all the information.

There are several factors which may influence the effects of long-sentence translations. According to our observation, we find that although some of our translation does not match the ground truth, the translation still make sense. We have randomly sampled 100 wrong translations, and categorised them into different types. Among the 100 sampled sentences, 52 of them are wrong translations with major problems including missing words, repeating words or translating words incorrectly. Either the meaning or structure of these translations are twisted from the ground truth, which means they are considered failure translations of the model. The remaining 42 sentences can serve as eligible alternatives for the expected translations. 1) the voice of our translation is different from that of target. For example, there is one sentence in our training corpus "your schedule has been saved successfully!". This sentence is translated to 已成功保存您的计划！ (active voice) by our model. However, the ground truth is 您的计划已被成功保存！ (passive voice). The changed voice sentences consists of 22.9%(11 out of 48) of this type of sentences. 2) the synonyms and abbreviations make our translation different from ground truth, though they are both right. For example, the ground truth translation of "failed to get last SMS sender" is "无法得到最后短信发送者", while it is translated to "未能获取最后短信的发件人" by our model. In this example, the pairs (无法, 未能, failed), (得到, 获取, get), (发送者, 发件人, sender) are of the same meaning but expressed in different forms. We have counted the translation of word "sender", and find that 94 out of 146 prefer "发件人", while only 52 out of 146 use "发送者". Among the 48 wrong translations that make sense, 37 of them are due to using synonyms (77.1 %). Therefore, the exact match rate of our model actually is highly underestimated especially for long sentences.

However, for software localization tasks, software text are usually short sentences. Sentences longer than 30 words account for only a very small portion (about 0.8% of our translation corpus).

5.3 RQ2: Translation Across App Categories

5.3.1 Evaluation Setup

We train our translation model based on a translation corpus of the collected corpus. To check that if our trained translation model can be adopted for other Android applications, we randomly select another 40 apps that are not in our dataset. For these 40 apps, 20 of them are popular with at least 1 million installations, and the other 20 apps are unpopular with installation number less than 10,000. Note all of these apps contain more than 100 translation pairs, and they belong to 17 categories. These applications cover a wide range of application domains, including social (*Facebook*), music and audio (*SoundCloud*), media and video (*Youtube*, *Yarn*), communication (*Gmail*, *Firefox*), tools (*Google Play*, *Boomerang*), lifestyle (*Uber*), and business (*Amazon*).

5.3.2 Results

Table 6 summarizes the information of the selected 20 popular apps and the translation results. We calculate the BLEU score and exact match rate by comparing the translation results using our model or Google Translate with the human reference translations. The average BLEU score is 50.434% (slightly higher than the average BLEU score (49.51%) of translation test data), and the average exact match rate is 0.564 (much higher than the average exact match rate (0.471) of Android translation test data) because that the popular apps always contain more standard translation than other unpopular apps. These results demonstrate the increased performance of our translator. In terms of BLEU score, Instagram yields the best result (69.92%), while the worst result is Google Play (23.14%). The reason might be that Google Play contains the most sentences (4622) than other applications and a great proportion of them are very long. According to Fig. 10, sentence length have substantially effects on BLEU score. In terms of exact match rate, the best and worst results are also Instagram (0.7) and Google Play (0.44). This is because Instagram contains more frequent sentences and phrases than other applications.

For unpopular apps, we do not see any big difference from popular apps. The average BLEU score and exact match rate for unpopular apps is 51.06%, 0.567, while 50.434%, 0.564 for popular apps. In contrast, Google Translate greatly varies a lot when considering popular apps (0.322 exact match) and unpopular ones (0.273 exact match). Such results demonstrate that our model is robust to no matter popular apps, unpopular ones or in different categories (Table 7).

5.4 RQ3: Generalization Across other Languages

5.4.1 Evaluation Setup

We have demonstrated the our model works well in English-Chinese app localization. Apart from English, there are another 5 official languages used in United Nations (2018b) including Arabic, Chinese, French, Spanish and Russian. As we are highly concerned with software localization, we report the performance of our model in translating English into the other 5 languages in this section. The results are presented in Table 8. In addition, we also carry out the experiment about software internalization i.e. translating text from other languages to English, to further demonstrate the generalization of our model.

5.4.2 Results

Table 8 shows the BLEU score for the google translate and our model for translation between English and the other five UN official languages (Arabic, Chinese, French, Spanish, Russian).

In terms of BLEU score, our model outperforms Google Translate in all five languages. The best BLEU score across all five languages achieved by our model is Chinese, which is 49.51%, while French has the lowest BLEU score (25.61%). But compared with Google Translate, our model outperforms Google Translate in all five language-pair translations. Our model has the biggest boost than Google Translate in translating English to Russian, from 17.77% to 33.01% which means 85.8% increase. The smallest boost is to translate from English to Arabic from 25.67% to 28.95%, but it is still 12.8% increase. The average 31.3% performance boost demonstrate the generalization of our model to different datasets.

Table 6 Localization experiment on popular apps in different categories

Application	Category	#Installations	#Sentences	BLEU %				Exact match			
				C-2RNN	2RNN	GT	Phrasal	C-2RNN	2RNN	GT	Phrasal
Amazon	Business	100,000,000+	807	55.55	46.63	42.35	46.41	0.66	0.41	0.35	0.57
Booking	Travel & local	100,000,000+	2493	49	38.66	40.15	36.78	0.56	0.39	0.36	0.41
Boomerang	Tools	50,000,000+	170	56.06	44.72	46.65	45.01	0.69	0.49	0.42	0.58
Chrome	Communication	1,000,000,000+	1235	40.19	29.81	37.65	28.27	0.56	0.31	0.4	0.3
Didi mobility	Transportation	1,000,000+	3655	52.64	44.09	45.26	43.29	0.51	0.29	0.27	0.0.2
Duolingo	Education	100,000,000+	713	61.24	50.68	49.63	39.34	0.55	0.33	0.25	0.26
ESPN	Sports	10,000,000+	451	53.81	36.64	36.5	38.1	0.47	0.32	0.29	0.38
Facebook	Social	1,000,000,000+	1168	47.43	38.47	39.45	37.72	0.58	0.34	0.33	0.48
Firefox	Communication	100,000,000+	562	42.15	36.25	47.61	35.83	0.45	0.26	0.33	0.34
Gmail	Communication	1,000,000,000+	1860	53.64	39.87	41.81	43.76	0.59	0.34	0.35	0.51
Google Play	Tools	1,000,000,000+	4622	23.14	17.96	18.05	27.56	0.44	0.11	0.18	0.29
Instagram	Social	1,000,000,000+	1339	69.92	48.39	55.1	47.26	0.7	0.34	0.27	0.28
Microsoft Word	Tools	500,000,000+	321	50.06	42.08	49.85	24.24	0.51	0.23	0.26	0.35
Netflix	Entertainment	500,000,000+	1269	53.28	43.52	42.37	38.04	0.58	0.4	0.39	0.45
QQ	Commiunication	10,000,000+	3421	59.48	50.38	55.23	53.32	0.59	0.31	0.3	0.19
SoundCloud	Music & audio	100,000,000+	206	51.4	34.53	31.88	57.25	0.6	0.35	0.33	0.53
Twitter	News & magazines	10,000,000+	2638	45.15	40.2	44.91	42.12	0.49	0.35	0.42	0.39
Uber	Lifestyle	100,000,000+	936	51.54	39.26	41.58	47.94	0.63	0.36	0.35	0.55
Yarn	Media & video	5,000,000+	108	40.97	35.41	37.76	47.06	0.6	0.31	0.28	0.49
Youtube	Media & video	1,000,000,000+	1452	52.03	40.35	42.4	34.41	0.52	0.34	0.31	0.43
Average			1307	50.434	39.89	42.31	40.69	0.564	0.329	0.322	0.399

Table 7 Localization experiment on unpopular apps in different categories

Application	Category	#Installations	#Sentences	BLEU %				Exact match			
				C-2RNN	2RNN	GT	Phrasal	C-2RNN	2RNN	GT	Phrasal
Accession communicator	Communication	10,000+	1274	47.06	29.25	33.1	33.9	0.43	0.18	0.19	0.31
Black & White sportclub	Sports	500+	1240	52.6	45.27	44.8	45.6	0.64	0.27	0.27	0.37
Capsule Docs	Business	1,000+	1582	61.99	48.89	53.45	41.66	0.67	0.32	0.33	0.43
Carolina panthers news	Sports	10,000+	1265	47.64	43.84	42.03	46.21	0.78	0.35	0.27	0.45
DTS Play-Fi	Business	10,000+	1160	51.85	40.93	41.71	29.13	0.52	0.36	0.31	0.24
DJI forum	Photography	10,000+	1210	52.71	46.37	44.3	29.02	0.63	0.35	0.29	0.26
CWB communicator	Productivity	1,000+	965	50.35	35.18	36.12	34.5	0.45	0.24	0.21	0.33
Gaming community	Social	1,000+	1296	51.46	40.01	42	51.9	0.62	0.3	0.27	0.46
Haus forum	Social	10,000+	1240	52.64	39.51	44.72	46.85	0.62	0.26	0.28	0.38
Jive	Productivity	10,000+	1095	49.39	45.79	53.65	39.23	0.46	0.28	0.29	0.36
KSPolice	Finance	1,000+	539	54.82	48.21	49.05	42.02	0.56	0.29	0.33	0.48
myREINspace	Business	500+	1210	51.61	45.63	44.3	46.62	0.6	0.31	0.29	0.46
OneCoin	Finance	1,000+	540	53.2	50.69	50.13	45.69	0.55	0.36	0.34	0.49
Queens college mobile	Education	10,000+	1060	43.56	37.64	39.26	40.19	0.42	0.24	0.26	0.42
Retrofit me	Health	5,000+	1072	50.15	35.96	35.16	33.42	0.47	0.21	0.2	0.32
Seguridad wireless	Communication	10,000+	1210	51.65	45.24	44.27	47.91	0.59	0.28	0.28	0.49
Sinocom	Business	10,000+	1240	51.31	42.94	44.84	46.36	0.6	0.24	0.27	0.39
Talk B.V.	Social	1,000+	1336	55.6	41.11	44.77	44.9	0.62	0.25	0.27	0.4
Telerific	Communication	500+	1274	47.06	36.35	33.61	33.62	0.61	0.28	0.25	0.31
Western	Education	10,000+	970	44.62	36.28	37.43	40.85	0.49	0.29	0.26	0.41
Average			1138	51.06	41.75	42.93	40.979	0.567	0.283	0.273	0.388

Table 8 Translation result on 5 official languages of UN (BLEU Score)

Target	#Sentence pairs	C-2RNN	2RNN	GoogleTranslate
Arabic	84,294	28.95	24.80	25.67
Chinese	58,503	49.51	38.12	40.42
French	113,149	25.61	20.15	23.48
Spanish	117,795	35.97	28.37	28.54
Russian	89,958	33.01	18.45	17.77

For software internationalization, the experiment results can be seen in Table 9. The results shows that the performance of 2RNN is similar to that of Google Translate. But, our customized model (C-2RNN) still outperforms the other two baselines in the experiment. In addition, the Bleu score for using our model for localization and internationalization respectively is similar. It further demonstrates the robustness and generality of our model, to some extent.

5.5 RQ4: Usefulness of our Translation Model

5.5.1 Evaluation Setup

While the exact rate of our translation model is not perfect and cannot be directly applied in practice, we still believe that human translators may save time and improve translation quality if provided with translations generated by our model. To show the usefulness of our tool, we carry out an experiment to compare the working efficiency of translators with our tool and without our tool. We first randomly select 8 apps which are in the experiment of Section 5.3. For each of them, we randomly take 50 sentences for translations. We recruit two participants (T1, T2) who are Chinese master students in computer science with very similar English capability. Their latest IELTS (International English Language Testing System) scores are all 6.5, and they are all recognized as certified translators by NAATI (National Accreditation Authority of Translators and Interpreters) in Australia. They are assigned with these English text with the background of this app i.e., functionality, category, etc. Participant T1 will be given translations generated by our model, while participant T2 translate the text from scratch. They are allowed to use tools such as dictionaries (both paper-based and electronic) for looking up unfamiliar words while translating. We then measure how much time it takes to finish the translation and we compare the results with the ground truth. All the translation text are then forwarded to another PhD student who has experience in software localization to judge the x of the translation with marking as 5-point likert score (1 as completely unsatisfied and 5 as perfect).

Table 9 Translation result on 5 official languages of UN (BLEU Score)

Target	#Sentence pairs	C-2RNN	2RNN	GoogleTranslate
Arabic	84,294	27.31	20.16	24.73
Chinese	58,503	48.69	42.06	41.98
French	113,149	25.50	23.60	23.39
Spanish	117,795	33.12	26.99	28.26
Russian	89,958	32.58	21.83	20.49

5.5.2 Results

Figure 11 presents the results of our experiments. The average 5-point likert score of T1's translation without our tool is 3.59, while T2 achieves 4.07 (Fig. 11a). In terms of translation time, the average time used to translate 50 sentences for T1 is 815.5s, while the number for T2 is only 424s (Fig. 11b). These results indicate that the translator provided with translations generated by our model can obtain 13.3% performance boost with using only half of translation time. We have conducted paired two-tailed tests (Rice 1989) to compare translation quality and time of translator T1 and T2. The p-values of both experiment results (translation quality and time) is lower than 0.01 which indicates that our conclusion is significant. Therefore, these results demonstrate the usefulness of our model for assisting text translation in software localization.

6 Threats to Validity

6.1 Amount and Quality of Training Data

Deep learning techniques like RNN require a large amount of data for model training. We have collected more than half million bilingual sentence pairs. Although not comparable to the amount of data used for training general machine translation models, it is sufficient to train a prototype model which can outperform the general machine translation, Google Translate.

All translations in our dataset are contributed by thousands of apps on the Google Play platform. Some apps may be of poor quality and that may further lead to unstable translation quality. But on Google Play, many apps are of commercial purpose, and the software localization may determine if they can conquer more markets in the world. This motivation guarantees the translation quality to some extent.

Note that these translations from apps may not all be human translations. Instead, some of them may be automatically translated by some tools. To check if the collected parallel corpus are from human translators, we randomly sample 50 apps for manual checking. These 50 apps are diverse, covering 17 categories including finance, communication, tools,

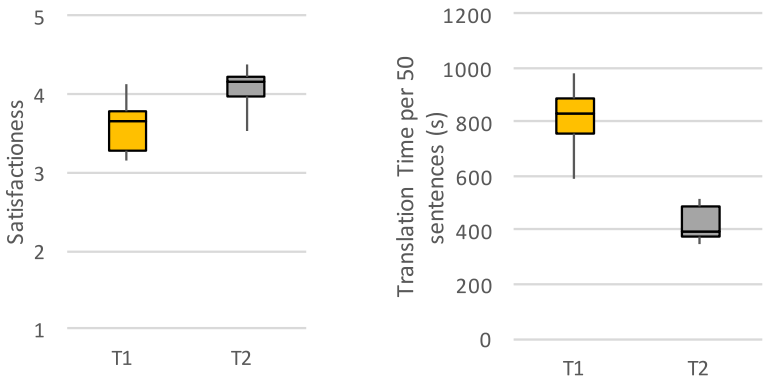


Fig. 11 Translation quality and time of two translators

etc with installation number from 10,000 to 10,000,000, and rating score from 3 to 5. There are 41 free apps and 9 apps with in-app purchases among the 50 apps. As we are only familiar with English and Chinese, the first author in this paper manually checks if the Chinese translation from English text is from human or machine. To help judge the translation source, he uses online translation tools including Google Translate[11] and Youdao Translator.[12] The results show that among 50 apps, no app solely uses online machine translation tools.[13] Although some translations are very similar to the results from online machine translation tools, we can still see some modification from human translators such as word order changes, tense difference, etc. For example, in an app the sentence "please give us five star" is translated to "请来支持我们吧" (please support us), while the direct translation (as well as translated by Google Translate) should be "请给我们五颗星". From this pilot study, it shows that to some extent, the collected corpus are from human translators or at least modified by human translators. Maybe some translations are from translation tools, but the experiment shows that most translation data collected in this work is from human translators or at least edited by human translators.

Although translations from different apps may not be the same for the same sentence, they can increase the diversity of the training data to make our model more robust. In the future, we will try to collect more dataset from different resources e.g., IOS or website to train our model.

6.2 Parameter Tuning

Deep learning algorithms are computation-extensive and our large-scale dataset makes the training process rather long. In this work, we do not experiment many different parameter settings due to the time and computation constraint, but we select the model parameters according to our previous experience Chen et al.(2016a, c). In the future, we will train our model with more powerful machines equipped with more GPUs to speed up the training process and try different parameter combinations to better tune our translation model.

7 Related Work

In this section, we first review research progress in software localization. Then we describe related work for machine translation, especially domain-specific machine translation. Finally, we introduce other research work that applys machine translation methods in software engineering tasks.

7.1 Software Localization

Among the 7.4 billion people in the world,[14] only 330 to 360 million people[15] speak English. To make people all over the world have access to a software product or service,

[11] https://translate.google.com/
[12] http://fanyi.youdao.com/
[13] The detailed checking results can be found in https://sites.google.com/view/domainspecifictranslation/
[14] https://en.wikipedia.org/wiki/World_population
[15] https://en.wikipedia.org/wiki/English-speaking_world

internationalization and localization of the software are necessary. Localization is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translating visible text of software.

At the early stage of development, developers may not consider the localization process due to various reasons. Therefore, many methods have been proposed by researchers to help developers spot code which may need to analyze (Xia et al. 2013), locate need-to-translate constant strings (Wang et al. 2010), and extract text (Rich 2011) from the software for translation. After translation, the old UI layout may not be suitable for the translated text in the new language. Some researchers carry out studies about how to identify such issues (Alameer et al. 2016) and automatically adjust the UI elements (Fitzpatrick et al. 2013; Burukhin et al. 2007). Different from these works which focus on pre-translation and post-translation steps in the process of software localization, our work focus on the core and the most labor-intensive step of software localization, i.e., the translation of software text. By complementing existing techniques with our work, software localization process can be better automated.

It is worth to see how industry work with software localization. To enhance software localization process, many software localization management systems are developed such as phraseapp (2018c), Smartling (2018d), Transifex (2018e), etc. All of these tools provide software localization platforms for supporting collaboration between developers and translators, product management, automated workflow for maintenance. However, they still require translators to manually translate the software text and put the translation into the platform instead of directly generating the translation results. Our domain-specific translator can be incorporated into the existing software localization tool to better support the software localization process.

7.2 Machine Translation

Machine Translation is an automated translation of text without human involvement. There are two main-stream methods for machine translation. The first type is based on the statistics (mostly phrase-based (Koehn et al. 2003; Zens et al. 2002)), called statistical machine translation. However, this type of methods always require many preprocessing such as sentence segmentation (Chung and Gildea 2009), phrase alignment (Fraser and Marcu 2007) and so on. In addition, the performance of statistical machine translation decreases significantly, in the face of new sentence patterns which do not appear in the training corpus. To overcome these shortcomings, the other kind of machine translation is based on neural networks called neural machine translation. In our work, we also adopt the state-of-the-art neural machine translation model, the RNN encoder-decoder model (Cho et al. 2014; Sutskever et al. 2014).

However, as general machine translation is always not effective for domain-specific text, it is necessary to develop domain-specific translation models. Due to the lack of domain-specific training data, most domain-specific machine translation approaches targets at domain adaptation of general machine translation models. Most of them (Wu et al. 2008; Ren et al. 2009; Zhang et al. 2013; Eck et al. 2004) post-process the translation results from general machine translation by referring to some domain-specific term dictionary or glossary. However, the required bilingual domain-specific term glossary is hard to build for software applications and it may be soon out-of-date because of the rapid development of application domains. Furthermore, considering some unique characteristics of software

text translation, e.g., some software names do not need to be translated, we customize the original neural machine translation with a special word alignment mechanism to handle out-of-vocabulary words.

7.3 Machine Translation in Software Engineering

Although machine translation has not been used for software localization, it has been adopted into software engineering domain for some other translation tasks such as translating the original posts into the edited posts in Stack Overflow (Chen et al. 2017a), translating the mobile UI into the source code (Chen et al. 2018b), translating the method description into the method name in Java (Gao et al. 2019), or translating the Chinese queries to English questions (Chen et al. 2016c). Nguyen et al. (2013, 2014, 2015) adopt statistical machine translation for language migration especially between *java* and *c#*. Oda et al. (2015) use phrase-based machine translation to automatically generate pseudo-code from source code. Gu et al. (2016, 2017) formulate pairs of natural-language query and corresponding API sequence as a machine translation problem so that developers can search API sequence by the natural-language query. And, they further put the code comments as a bridge between API sequences across different language to support software migration. Tjalling (2016) and Hu et al. (2018) use the similar neural machine translation method to generate comments from the code. Gu et al. (2018) have developed a code search tool for assisting code searching and reusing code for developers with deep neural network. Different from existing work, we apply neural machine translation model for the purpose of domain-specific natural-language translation for software localization.

8 Conclusion and Future Work

This work targets at an important development activity in software engineering, i.e., software localization, especially app translation. Specially, we focus on domain-specific machine translation for software localization, which is the core step for the success of software localization but receives little attention from the community. Our empirical study demonstrates that general machine translation models is not suitable for the translation of software text which has its own linguistic and technical characteristics. In this paper, we present a RNN-based domain-specific machine translation model. Considering the characteristics of software text, we customized the original RNN encoder-decoder model by addressing domain-specific rare word problem and training the model using a large-scale domain-specific translation corpus collected from the apps in Google Play. The results of experiments demonstrate the effectiveness of our model, compared with the general machine translation, Google Translate.

In the future, we will continue to improve the model by tuning its parameters and training it on larger dataset. In addition to Android app localization, there are no intrinsic barriers to apply our model to more platforms for software localization, such as IOS, website, desktop software, etc. We will incorporate our method into other research works such as locating need-to-translate text (Wang et al. 2010) and adjusting the UI layout after text translation (Burukhin et al. 2007) to better automate software localization process. Another interesting direction is to apply our translation method into other software engineering tasks which could be formulated as translation problems, such as application migration across programming languages, analogical libraries recommendation, and pesudo-code generation.

References

Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. In: ACM Sigmod Record, ACM, vol 22, pp 207–216

Alameer A, Mahajan S, Halfond WG (2016) Detecting and localizing internationalization presentation failures in web applications

Alshaikh Z, Mostafa S, Wang X, He S (2015) A empirical study on the status of software localization in open source projects

Apktool (2018) A tool for reverse engineering android apk files. https://ibotpeaches.github.io/Apktool/

Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. arXiv:14090473

Borgelt C (2012) Frequent item set mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(6):437–456

Burukhin A, Gadre MA, Aldahleh AM, Farrell T, Larrinaga-Pardo JL (2007) Dynamically providing a localized user interface language resource. US Patent App. 11/869,083

Chen C, Chen X, Sun J, Xing Z, Li G (2018a) Data-driven proactive policy assurance of post quality in community q&a sites. Proceedings of the ACM on human-computer interaction 2(CSCW):33

Chen C, Gao S, Xing Z (2016a) Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), vol 1. IEEE, pp 338–348

Chen C, Su T, Meng G, Xing Z, Liu Y (2018b) From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation. In: Proceedings of the 40th international conference on software engineering. ACM, pp 665–676

Chen C, Xing Z (2016a) Mining technology landscape from stack overflow. In: Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement. ACM, p 14

Chen C, Xing Z (2016b) Similartech: automatically recommend analogical libraries across different programming languages. In: 2016 31st IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 834–839

Chen C, Xing Z, Han L (2016b) Techland: assisting technology landscape inquiries with insights from stack overflow. In: 2016 IEEE international conference on software maintenance and evolution (ICSME). IEEE, pp 356–366

Chen C, Xing Z, Liu Y (2017a) By the community & for the community: a deep learning approach to assist collaborative editing in q&a sites. Proceedings of the ACM on Human-Computer Interaction 1(CSCW):32

Chen C, Xing Z, Liu Y (2018c) What's spain's paris? mining analogical libraries from q&a discussions. Empir Softw Eng, pp 1–40

Chen C, Xing Z, Liu Y, Ong KLX (2019) Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding. IEEE Trans Softw Eng

Chen C, Xing Z, Wang X (2017b) Unsupervised software-specific morphological forms inference from informal discussions. In: Proceedings of the 39th international conference on software engineering. IEEE Press, pp 450–461

Chen G, Chen C, Xing Z, Bowen X (2016c) Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In: 31st IEEE/ACM international conference on automated software engineering (ASE), IEEE/ACM

Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv:14061078

Chung T, Gildea D (2009) Unsupervised tokenization for machine translation. In: Proceedings of the 2009 conference on empirical methods in natural language processing: Volume 2-Volume 2, Association for computational linguistics, pp 718–726

Eck M, Vogel S, Waibel A (2004) Improving statistical machine translation in the medical domain using the unified medical language system. In: Proceedings of the 20th international conference on computational linguistics, association for computational linguistics, p 792

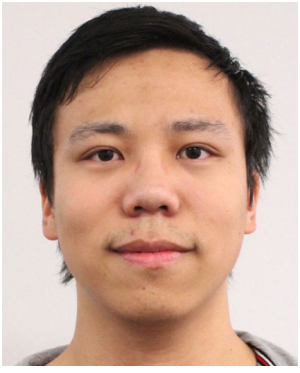
Fitzpatrick C, Whelan JP, Doyle RP, Lane JG, McHugh B, Farrell T, Barnes P, McQuaid AM, Mowatt D (2013) Dynamic screentip language translation. US Patent 8,612,893

Fraser A, Marcu D (2007) Measuring word alignment quality for statistical machine translation. Comput Linguist 33(3):293–303

Gao S, Chen C, Xing Z, Ma Y, Song W, Lin SW (2019) A neural model for method name generation from functional description. In: 2019 IEEE 26th international conference on software analysis, evolution, and reengineering (SANER), vol 1. IEEE

Graves A, Mohamed A-r, Hinton G (2013) Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, pp 6645–6649
Green S, Cer D, Manning C (2014) Phrasal: a toolkit for new directions in statistical machine translation. In: Proceedings of the ninth workshop on statistical machine translation, pp 114–121
Google Play Store (2018a). https://play.google.com/store
Gu X, Zhang H, Kim S (2018) Deep code search. In: Proceedings of the 40th international conference on software engineering. ACM, pp 933–944
Gu X, Zhang H, Zhang D, Kim S (2016) Deep api learning. arXiv:160508535
Gu X, Zhang H, Zhang D, Kim S (2017) Deepam: migrate apis with multi-modal sequence to sequence learning. arXiv:170407734
Holzer H, Ant F, Nogueira D, Semolini K, Martin C, Aiken M, Balan S, Zetzsche J, Avval SF, Carl M et al (2011) An analysis of google translate accuracy
Hu X, Li G, Xia X, Lo D, Jin Z (2018) Deep code comment generation. In: Proceedings of the 26th conference on program comprehension. ACM, pp 200–210
Huang Y, Chen C, Xing Z, Lin T, Liu Y (2018) Tell them apart: distilling technology differences from crowd-scale comparison discussions. In: Proceedings of the 33rd ACM/IEEE international conference on automated software engineering. ACM, pp 214–224
Koehn P, Och FJ, Marcu D (2003) Statistical phrase-based translation. In: Proceedings of the 2003 conference of the North American chapter of the association for computational linguistics on human language technology-volume 1, association for computational linguistics, pp 48–54
Luong MT, Sutskever I, Le QV, Vinyals O, Zaremba W (2014) Addressing the rare word problem in neural machine translation. arXiv:14108206
Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv:13013781
Mikolov T, Dean J (2013) Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems
Mikolov T, Deoras A, Povey D, Burget L, Cernockỳ J (2011) Strategies for training large scale neural network language models. In: 2011 IEEE workshop on automatic speech recognition and understanding (ASRU). IEEE, pp 196–201
Mikolov T, Karafiát M, Burget L, Cernockỳ J, Khudanpur S (2010) Recurrent neural network based language model. In: Interspeech, vol 2, p 3
Muntés Mulero V, Paladini Adell P, España Bonet C, Màrquez Villodre L (2012) Context-aware machine translation for software localization. In: Proceedings of the 16th annual conference of the European association for machine translation: EAMT 2012: Trento, Italy, May 28th-30th 2012, pp 77-80
United Nations (2018b) http://www.un.org/en/sections/about-un/official-languages/index.html. http://ask.un.org/faq/14463, Accessed 2018-06-20
Nguyen AT, Nguyen TT, Nguyen TN (2013) Lexical statistical machine translation for language migration. In: Proceedings of the 2013 9th joint meeting on foundations of software engineering. ACM, pp 651–654
Nguyen AT, Nguyen TT, Nguyen TN (2014) Migrating code with statistical machine translation. In: Companion proceedings of the 36th international conference on software engineering. ACM, pp 544–547
Nguyen AT, Nguyen TT, Nguyen TN (2015) Divide-and-conquer approach for multi-phase statistical migration for source code (t). In: 2015 30th IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 585–596
O'Brien S (1998) Practical experience of computer-aided translation tools in the software localization industry. Unity in diversity pp 115–122
Oda Y, Fudaba H, Neubig G, Hata H, Sakti S, Toda T, Nakamura S (2015) Learning to generate pseudo-code from source code using statistical machine translation (t). In: 2015 30th IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 574–584
Papineni K, Roukos S, Ward T, Zhu WJ (2002) Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting on association for computational linguistics, association for computational linguistics, pp 311–318
Phraseapp (2018c) Software translation management. https://phraseapp.com/, Accessed 2018-06-20
Plamada M, Volk M (2013) Mining for domain-specific parallel text from wikipedia. ACL 2013, pp 112
Ren Z, Lü Y, Cao J, Liu Q, Huang Y (2009) Improving statistical machine translation using domain bilingual multiword expressions. In: Proceedings of the workshop on multiword expressions: identification, interpretation, disambiguation and applications, association for computational linguistics, pp 47–54
Rice WR (1989) Analyzing tables of statistical tests. Evolution 43(1):223–225
Rich DP (2011) Method and system for improved software localization. US Patent 7,987,087
Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620

Smartling (2018d) Smartling global content translation and localization solution. https://www.smartling.com/, Accessed 2018-06-20
Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp 3104–3112
Tjalling H (2016) Automatic comment generation using a neural translation model
Transifex (2018e) Transifex: Localization platform for translating digital content. https://www.transifex.com/, Accessed 2018-07-20
Turian J, Ratinov L, Bengio Y (2010) Word representations: a simple and general method for semi-supervised learning. In: Proceedings of the 48th annual meeting of the association for computational linguistics, association for computational linguistics, pp 384–394
Wang X, Zhang L, Xie T, Mei H, Sun J (2010) Locating need-to-translate constant strings in web applications. In: Proceedings of the eighteenth ACM SIGSOFT international symposium on foundations of software engineering. ACM, pp 87–96
Werbos PJ (1990) Backpropagation through time: what it does and how to do it. Proc IEEE 78(10):1550–1560
White M, Vendome C, Linares-Vásquez M, Poshyvanyk D (2015) Toward deep learning software repositories. In: 2015 IEEE/ACM 12th working conference on mining software repositories (MSR). IEEE, pp 334–345
Wu H, Wang H, Zong C (2008) Domain adaptation for statistical machine translation with domain dictionary and monolingual corpora. In: Proceedings of the 22nd international conference on computational linguistics-volume 1, association for computational linguistics, pp 993–1000
Wu Y, Schuster M, Chen Z, Le QV, Norouzi M, Macherey W, Krikun M, Cao Y, Gao Q, Macherey K, et al. (2016) Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:160908144
Xia X, Lo D, Zhu F, Wang X, Zhou B (2013) Software internationalization and localization: an industrial experience. In: 2013 18th international conference on Engineering of complex computer systems (ICECCS). IEEE, pp 222–231
Zens R, Och FJ, Ney H (2002) Phrase-based statistical machine translation. In: Annual conference on artificial intelligence. Springer, pp 18–32
Zhang J, Zong C et al (2013) Learning a phrase-based translation model from monolingual data with application to domain adaptation. In: ACL, vol 1, pp 1425–1434



Xu Wang is a graduate of Australian National University, Australia. He obtained the Master of Computing (Advanced) degree from ANU in 2018. Xu's research area focuses on applying deep learning techniques to solve software engineering problems.

Chunyang Chen is a lecturer in the Faculty of Information Technology, Monash University, Australia. He obtained the Ph.D degree from Nanyang Technological University, Singapore in 2018. Dr. Chen's research mainly focuses on Software Engineering and Human-computer Interaction. His works are mostly about mining software repositories to distill insights from big data. Based on such insights, he develops many tools to assist developers in the software development process. Apart from developers, he also carries out research about assisting designers with the UI design.

Zhenchang Xing is the senior lecturer at the research school of computer science, Australian National University, Australia. Dr. Xing's research interests include software engineering and human-computer interaction. His work combines software analytics, behavioral research methods, data mining techniques, and interaction design to understand how developers work, and then build recommendation or exploratory search systems for the timely or serendipitous discovery of the needed information.

Affiliations

Xu Wang[1] · Chunyang Chen[2] · Zhenchang Xing[1]

Xu Wang
u5833088@anu.edu.au

Zhenchang Xing
zhenchang.xing@anu.edu.au

[1] College of Engineering & Computer Science, Australian National University, Canberra, Australia

[2] Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia