Statistical Machine Translation of English Text to API Code Usages:

A comparison of Word Map, Contextual Graph Ordering, Phrase-based, and Neural Network Translations

Dharani Kumar Palani

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

July 2018

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Dharani Kumar Palani**

Entitled: **Statistical Machine Translation of English Text to API Code Usages:**
A comparison of Word Map, Contextual Graph Ordering, Phrase-based, and Neural Network Translations

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair

_____ Examiner
Dr. Juergen Rilling

_____ Examiner
Dr. Rene Witte

_____ Supervisor
Dr. Peter C. Rigby

Approved by _____
Dr Volker Haarslev, Graduate Program Director

30 July 2018 _____
Dr Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Statistical Machine Translation of English Text to API Code Usages:

A comparison of Word Map, Contextual Graph Ordering, Phrase-based, and Neural Network Translations

Dharani Kumar Palani

Statistical Machine Translation (SMT) has gained enormous popularity in recent years as natural language translations have become increasingly accurate. In this thesis we apply SMT techniques in the context of translating English descriptions of programming tasks to source code. We evaluate four existing approaches: maximum likelihood word maps, ContextualExpansion, phrase-based, and neural network translation. As a training and test (*i.e.* reference translation) data set we clean and align the popular developer discussion forum StackOverflow.

Our baseline approach, WordMapK, uses a simple maximum likelihood word map model which is then ordered using existing code usage graphs. The approach is quite effective, with a precision and recall of 20 and 50, respectively. Adding context to the word map model, ContextualExpansion, is able to increase the precision to 25 with a recall of 40. The traditional phrase-based translation model, Moses, achieves a similar precision and recall also incorporating the context of the input text by mapping English sequences to code sequences. The final approach is neural network translation, OpenNMT. While the median precision is 100 the recall is only 20. When manually examining the output of the neural translation, the code usages are very small and obvious. Our results represent an application of existing natural language strategies in the context of software engineering. We make our scripts, corpus, and reference translations in the hope that future work will adapt these techniques to further increase the quality of English to code statistical machine translation.

Acknowledgments

I would like to begin by acknowledging the guidance, encouragement, and support from my supervisor, Dr. Peter Rigby. Without his proper supervision, this work would not have come into existence. I wholeheartedly thank him for his patience, effort and time towards me and my thesis.

I am also thankful to my fellow colleagues and paper collaborators, especially Musfiqur Rahman and Nicolas Chausseau-Gaboriault.

I would like to thank Dr. Tien Nguyen (University of Texas at Dallas) for his time and guidance.

Last but certainly not least, I thank my wife, children, and parents who encouraged and supported me throughout the time of my research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software developers construct software systems by reusing the functionality contained in software libraries. These libraries allow access to their functionality through Application Programming Interfaces (API). Each API consists of set of named functionality in the form of named functions (*i.e.* actions) and classes (*i.e.* entities). Developers combine these named functions into API usages to accomplish a wide range of tasks. Since not not all API usages are recorded in the official usage documentation and programming guides, learning a new API is difficult [77, 73, 74]. To learn an API developers turn to web based search engines such as Google, Yahoo, and Bing. However web search engines are not designed for source code search and work based on the keywords found in the input text[81].

To recommend an API usage for an English query, earlier works use *Information Retrieval (IR)* techniques [84, 68, 76, 35]. However they are limited finding relevant code snippets instead of generating potentially unseen translations of the query to code. Statistical natural language translation[43] has been quite successful and has lead to commercial solutions, such as the translation engines developed by Google and Microsoft [87]. One key benefit of the statistical machine translation in natural languages is the ability to generate fluent translations with the use of a language model. Hence compared to that of IR methods, the statistical approaches can synthesize new unseen code.

Previous machine translation works in a software context have use word based alignment models (SWIM[69]), *Recurrent Neural Network (RNN)* (DeepAPI [30]), and context sensitive word mappings (T2API [61, 60]) to translate English text into software code. Previous research has focused on modifying a single SMT technique for English to code translation.

There are many exists, successful statistical machine translation systems. Our goal is to find the right statistical machine translation system to synthesize source code from English text. This is the first study to empirically evaluate multiple existing machine translation techniques on single set of

Table 1: Reference translation from StackOverflow discussion post 13761367. English words such as "GPS" map to related `Locaation` code elements.

Input text	**After NLP processing**	**Reference translation**
How to get Latitute Longitude value without using GPS in Android?	latitut longitud valu latitud gps devic network_provid	Location.Location Location.getLongitude LocationManager.getLastKnownLocation Location.getLatitude LocationManager.LocationManager Context.getSystemService LocationListener.onLocationChanged LocationListener.LocationListener LocationManager.isProviderEnabled LocationManager.requestLocationUpdates

reference translations. In our study we directly apply existing state of the art machine translation approaches to translate English text to API call usages. We compare the performance of each approach. We contrast our work with IR and suggest future work to modify SMT in the context of software engineering tasks.

To create a common training and test set, we use the popular StackOverflow form which contains discussions of software programming in both English and text. We clean and align StackOverflow to create a common training and test set, *i.e.* a reference translation. Table 1 is a reference translation, containing the input text, the text after stemming and other basic NLP processing, and the corresponding code elements that are in the StackOverflow post. The reference translation shows that the English words "without using GPS" should be related to the location API elements including `LocationManager.getLastKnownLocation()`.

We evaluate four existing machine translation techniques: WordMapK, ContextualExpansion, Phrase-based, and Neural Network translation. Our baseline approach, WordMapK, uses a simple maximum likelihood model to map the English input to K API code elements in the output. Since the K elements are an unordered set, we use Nguyen *et al.*'s [60] GraphOrdering to provide an appropriate sequence of elements. The precision and recall for WordMapK is 20 and 50, respectively.

The WordMapK model does not take the context of the English input into account. We use the ContextualExpansion approach [60] to incorporate commonly co-occurring code elements. A GraphOrdering is also used. The median precision and recall for ContextualExpansion is 25 and 40, respectively.

Maximum likelihood translation models are not used in current machine translation. As a result,

we evaluated existing phrase-based Moses tool on English to code translation. In phrase-based translation sequences of words are mapped to sequences of code. These sequences mean that the context of sequence is taken into account. Moses achieves a precision and recall of 25 and 40. While these values are the same as ContextualExpansion, a manual inspection of the translations shows that ContextualExpansion suggests more code elements than Moses.

Neural network translation is the current state-of-the-art. We use OpenNMT to evaluate how well neural translations can be applied to English to code translation. We find a median precision and recall of 100 and 20, respectively. Unfortunately, the output is usually limited to a very small and obvious API usage.

This thesis shows the results of using existing SMT approaches to translate English input to API calls sequences. We conclude that existing SMT approaches have low precision and recall when compared with other SE specific learning.

In order to test and measure the translation performance of machine translation systems, we develop a simple web interface. The web console accepts English text from the user and returns the translated API elements from each of the four SMT systems.[1] The tool can also be accessed programmatically through a REST API allowing others to send English input and obtain the translated output. This facilitates the development of code suggestion tools for *Integrated Development Environments (IDE)*. Our tool could also be combined with other translation systems and IR techniques to supplement the search output with potentially unseen code snippets suggestions.

This thesis is organized as follows. Chapter 2 discusses the mathematical background on the statistical machine translation systems including word based, phrase based and neural network based translation models. Chapter 3 presents the parallel corpus, SMT toolkits, and evaluation setup. Chapter 4 presents the results. Chapter 5 discusses our results as well as the computational cost of each approach, threats to validity and related work. Chapter 6 present the web portal. In Chapter 7 we provide brief closing remarks.

[1] CompareSMT: `https://users.encs.concordia.ca/~pcr/CompareSMT.html`

Chapter 2

Background

Machine translation has been in existence from the early 1940s. The cold war boosted work in this area as researchers looked for an automatic means of translation between English and Russian. The work in statistical machine translation was started in late 1980's in the IBM research laboratories. The techniques were borrowed from speech recognition. The initial intent was to train a machine with large parallel corpora of existing translation. There were two fundamental problems in the statistical machine translation:

- Translation of words.
- Order words after translation to give correct phrases.

Both problems are addressed by building mathematical models[16, 14, 15]

2.1 Translation Probability

Inferential statistics are implicitly used in all statistical machine translation techniques. A simple and intuitive model can be created by measuring the count of occurrence of words and their corresponding translations in a parallel corpora. The translation counts are used to build a lexical translation probability distribution by using the ratio of the counts. The probability distribution function that returns a translation probability for a given word s from the source language, the choice of word t in the target language which indicates how likely the translation is.

$$p_s : t \rightarrow p_s(t)$$

This function will return a high value when the probability of translating the word s to word t is quite common and a low value when such a translation is rare. The function returns 0 when there is

no translation. This method of estimation is called maximum likelihood estimation as it maximizes the likelihood for each translation.

2.2 Alignment Function

The translation probability function produces translation tables ("T-Tables") each word from the source language is mapped to possible outcomes in the target language. A translation system need to translate not just words but sentences. Sentences are translated using an alignment function which maps the word from the source language in position i to a target language word at position j. The alignments are defined as

$$a : j \rightarrow i \tag{1}$$

This alignment function is actually an inverse mapping function which map words in the target language to the words in the position i of the source language. The alignment function does not handle the following scenarios:

- Reordering - translation requires the words to be in different order.
- Addition of words - word in the source language requires more than one word in the target language.
- Deletion of words - There is no correspondence between the words in the source and target language.

To address the inclusion of words in the target language, a "NULL" token is added in the source language. The lexical translation probabilities and the alignment function defined above was implemented in the first statistical machine translation model namely the "IBM Model 1" [15]. The motivation for such a word based model is the difficulty in estimating the translation probabilities for whole sentences as their occurrence in the corpus may be limited. The process of breaking up the process of generating the data into smaller steps, modelling the smaller steps with probability distributions, and combining the steps is called generative modelling

The translation probability of translating a sentence $s = (s_1, s_2, \ldots, s_{ns})$ from the source language to the sentence in target language t = (t1, t2, ..., tnt) is defined in the IBM Model 1 as

$$p(t, a|s) = \frac{\epsilon}{(n_t + 1)^{n_s}} \prod_{j=1}^{n_t} p(t_j | s_{a_(j)}) \tag{2}$$

2.3 Expectation Maximization Algorithm

The previous section shows the translation probability when translating a sentence from source language to target language. However when the corpora is not word aligned we do not have the alignment probability function fully defined. The *Expectation Maximization (EM)* algorithm[25] solves the problem by iteratively training the model according to the following:

- Initialize the model, typically with uniform distributions.
- Apply the model to the data (expectation step).
- Learn the model from the data (maximization step).
- Iterate steps 2 and 3 until convergence.

Since the word alignment is unavailable apriori, one simple strategy is to use uniform probability distributions to initialize the model. This means that any word in a given source language sentence has an equiprobable translation for all words in the target language sentence. Other ways to initialize the model is to use random probabilities. In the expectation step, the model is applied to the data. In the maximization step, the model is learned from the data. The learning of model takes place by maximum likelihood estimation which uses the ratio of counts of the translation. The expectation and maximization steps are run until there is improvement in the translation probabilities. The EM algorithm has a guarantee that the perplexity of the model will not increase in every iteration and hence will ultimately converge.

2.3.1 Language Models

The language model helps in generating a better translation of the target language given an unknown set of words in the source language. The most common language model used is n-gram language model. The most common n-gram language model used is the 3-gram model, which takes the two previous tokens in predicting the next token. The estimation of 3-gram word prediction probabilities $p(w3|w1, w2)$ is predicted using a Markov Chain. We count how often in the training corpus the sequence $w1$, $w2$ is followed by the word $w3$, as opposed to other words. The maximum likelihood estimation is applied to the statistical data which computes probability of predicting the 3rd token given two previous tokens.

$$p(w3|w1, w2) = \frac{count(w1, w2, w3)}{\sum_w count(w1, w2, w)} \tag{3}$$

2.3.2 Noisy Channel Model

The Noisy channel model developed by Shannon [80] is used widely in speech recognition and information theory. This approach provides a way to decode messages sent through a noisy channel by reconstructing them using the language model. This analogy is applied in the statistical machine translation as well. The translated sentence from the statistical machine translation is assumed to have gone through a noisy channel which has added noise and it will be reconstructed correctly using the language model of the target language. It is expressed mathematically as

$$\arg\max_t p(t|s) = \arg\max_t \frac{p(s|t)pf(t)}{p(s)} = \arg\max_t p(s|t)p(t) \tag{4}$$

2.4 Phrase Based Models

Word based models face the problems of fertility, where one word in the source language maps to none or many words in the target language. Hence we need to add or remove words in the target language to achieve fluency, *i.e.* readable sentences. Also word based translation do not capture the surrounding context of the sentence. Phrase based statistical machine translation was developed to overcome these problems [43]. The benefit of phrase based models is that given a large corpora the model can learn and translate sequences of words, *i.e.* phrases. The phrases in the phrase based translation are just sequence of words. These phrases are statistically selected without any syntactic or other specific linguistic significance. Phrase based translation selects the best translation for a target language given the source language input sentence according to the following:

$$t_{best} = \arg\max_t p(t|s) = \arg\max_t p(s|t)p_{LM}(t) \tag{5}$$

The p(s|t) is further decomposed into

$$p(\bar{s}_{i=1}^I|\bar{t}_{i=1}^I) = \prod_{i=1}^I \phi(\bar{s}_i|\bar{t}_i)d(start_i - end_{i-1} - 1) \tag{6}$$

The phrases are reordered in the target language by applying a distance based reordering model. The reordering distance is the number of words skipped during translation. The reordering function is defined with a exponential decay cost function with a value α that makes the resulting function a proper probability distribution function.

$$d(x) = \alpha^{|x|} \tag{7}$$

The reordering function penalizes the movement of words over long distances. The phrase extraction process will look into the word alignment between the source and the target language,

find all possible phrases in the target language and finding the minimal source language phrase that matches each of them [43].The model is trained on phrases of arbitrary length. Moses by default build phrase tables with phrases of maximum length 7. Moses documentation suggests that the optimal number of words in a phrase is 3. For each sentence pair, multiple phrase pairs are extracted. Then a count of how many sentence pairs a particular phrase pair is present is computed and stored $count(\bar{t}, \bar{s})$. Finally, the phrase translation probability $\phi(\bar{t}, \bar{s})$ is estimated by the relative frequency.

$$\phi(\bar{s}|\bar{t}) = \frac{count(\bar{t}, \bar{s})}{\sum_{\bar{s}_i} count(\bar{t}, \bar{s}_i)} \tag{8}$$

The phrase based translation model is derived as

$$e_{best} = \arg\max_{t} \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i) d(start_i - end_{i-1} - 1) \prod_{i=1}^{|e|} P_{LM}(e_i|e_1 ... e_{i-1}) \tag{9}$$

This model consists of three components

- Phrase Translation Table - Foreign phrases matching the target language words
- Reordering Model - which helps to order the phrases
- Language Model - to help with fluency

2.5 Machine Translation using Neural Networks

Artificial Neural networks is a machine learning technique which is inspired by biological neural networks. They take several inputs and predict outputs. The first models for statistical machine translation using neural networks were derived during 1980's and 1990s. Neural network were limited in this era because of their computational complexity and the lack of computational resources. The modern neural network based machine translation was first done by Schwenk [79]. *Neural machine translation (NMT)* have increased in popularity, and in 2017 at the Conference on Machine translation all the machine translation models used neural networks. NMT differ from the phrase based translation systems in that there is a single large system that is used for learning and decoding, whereas in the phrase based translation system there are several components that requires separate tuning.

NMT is different from the phrase based machine translation system in that they use vector representation for words and internal states. There is a simple model and no separate models such

as language model, translation model and reordering model as observed in the phrase based or word based machine translation systems.

The most common approach used in the neural machine translation is encode-decoder architecture [82, 19, 32] with an encoder and a decoder used for each language which is called the *Recurrent Neural Networks (RNN)* Encoder-Decoder first proposed by [82] and [19]. In this framework, an encoder scans the input sentence $x = (x_1, ..., x_{T_x})$ and convert it into a vector. The transformation functions used are

$$h_t = f(x_t, h_{t-1}) \tag{10}$$

and

$$c = q(h_1, ..., h_{T_x}) \tag{11}$$

Here $h_t \epsilon \mathbb{R}^n$ is the hidden state at the time instance t and c is the vector obtained from the sequence of hidden states. The choice of functions (f and q) used depends on the type of the RNN Encoder-Decoder. In general a softmax function is used. The decoder is trained to predict the next word $y_{t'}$ given the context vector c and all the previously predicted words $y_1, ..., y_{t_0-1}$. This is the joint probability of the previously predicted words as given in the following equation

$$p(y) = \prod_{t=1}^{T} p(y_t | y_1, ..., y_{t-1}, c) \tag{12}$$

Where this conditional probability is modelled as given in the equation below

$$p(y_t, ..., y_{t-1}, c) = g(y_{t-1}, s_t, c) \tag{13}$$

The function g is the function which outputs the probability of y_t using the hidden state of the neural network represented in s_t and the context vector c.

Chapter 3

Methodology and Data

3.1 Transforming StackOverflow into a Parallel Corpus

StackOverflow is a forum where software developers post and answer programming questions [88, 1]. We treat StackOverflow as a bilingual corpus as it describes programming tasks in two languages: English and a programming language, *e.g.,* Java. For example, in Figure 1 we see the StackOverflow post #13761367 that asks the programming question, "How to get Latitude Longitude value without using GPS in Android?" In the answer post in Figure 2, the author describes in English how to use the code elements `LocationManager` and `Location.getLatitude()`.

StackOverflow is not a parallel corpus. Such a corpus would require a documents that contain the exact same content in two languages. For example, the proceedings of the Canadian Parliament includes a human generated transcript in both English an French. In previous works, we have documented and resolved much of the noise and lack of balance between code and English [70]. In this work, we provide an overview of the techniques. Our goal is to create a parallel corpus from StackOverflow which aligns the English parts of the post with a corpus that contains the code elements from the same post.

3.1.1 Automatically extracting code elements from posts

To build this parallel corpora, we process 236,919 posts from *StackOverflow (SO)* [61] using Rigby and Robillard's ACE tool [72]. ACE can extract code elements from freeform text and code snippets that do not necessarily compile. ACE identifies type and package information from code elements in freeform texts and incomplete code snippets. ACE extracts APIs embedded within texts. It removes stopwords such as (a, the) from the posts and extracts keywords/keyphrases (latitude, longitude, without. The parallel corpus was prepared using the collection of pairs in which each pair consists of a *textual*

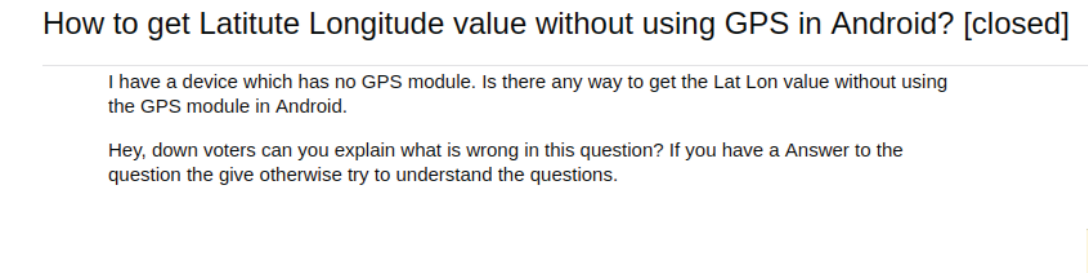


Figure 1: Example of StackOverflow question post #13761367

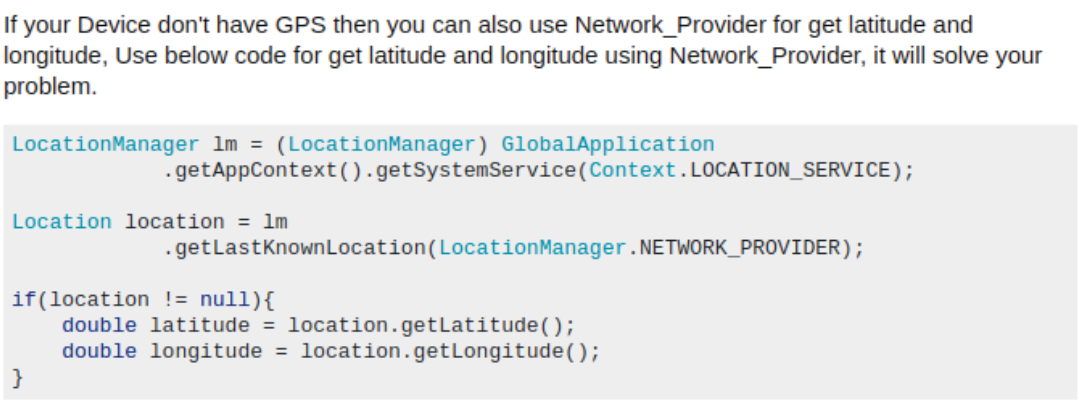


Figure 2: Answer post on StackOverflow #13761367

description (excluding the code elements in the text and the code snippet) and the set of *extracted code elements* from both places.

For example, in Figure 2 the extracted English terms include *"latitude longitude value", "without using the GPS", and "use Network Provider"*. The corresponding set of code elements include Location.Location, Location.getLongitude, LocationManager.getLastKnownLocation, and Location.getLatitude.

The English terms from each post are placed in the English corpus and the code elements are placed in the code corpus. The two corpora are aligned on the post. We measure the quality of each sentence pair (in our case English and code elements) by measuring their balance i.e number of words in English text to code elements. We ignore the SO posts which have an imbalance in the number of words to code elements. This parallel corpus is used to train each of the SMT techniques. The corpus can be obtained from our replication package [66].

3.1.2 Reference translations

A reference translation is a human generated parallel corpora used as a test set to evaluate the performance of machine translation systems. In the context of translating natural languages, a

human generated transcript is used. For example, to evaluate machine translation systems for translating natural language text from English to French and vice-versa, the human generated transcript containing the proceedings of the Canadian parliament [21] discussion in both English and French is used as a reference translation. Similarly proceedings from European parliament [41] is used as parallel corpus to test the performance of the machine translation systems for several European languages.

In our study we manually prepare a reference translation using English text and code elements from the StackOverflow posts. We use this parallel corpus as the test set to evaluate different SMT approaches in this paper. We randomly select 240 posts from StackOverflow to prepare this parallel corpora. To prepare English text in the parallel corpora, we take text from the title (i.e question) and discussions surrounding the positively voted and accepted answers. We apply NLP techniques to the English text such as *stemming* and *stop word removal*. Table 2 shows query after the stemming and stop word removal steps.

To prepare code elements in the parallel corpora, we extract code elements from the positively voted and accepted answers in the SO post using *ACE Tool* and *RECODER*. For some StackOverflow posts, we manually curate the code elements by referring the source code in the positively voted and accepted answers. This is because in some SO posts, code elements are described in the text and code snippets might not represent a solution in entirety.

Figures 1 and 2 is one of the SO post in our reference translation corpora. Table 2 gives the details of the input text used in the post and the code elements taken from the positively voted and accepted answers. The reference translation contains the key classes and methods that needs to be invoked to retrieve the current location without the use of GPS provider.

As we have discussed in previous work, StackOverflow requires substantial cleaning for training an SMT model [70]. Unlike this training set, the reference translation in SMT is normally manually created. As a result, we use the a random sample of 240 posts that were manually validated as our reference translation set. The reference translations can be found in our replication package [66].

3.1.3 Tuning Set

The machine translation systems requires one additional parallel corpora to tune the parameters and find the optimized weights of the trained models. This parallel corpora is called *tuning set* or *validation set*. The validation set is used to evaluate the convergence of the training. The trained model that achieves lowest perplexity on the validation set is in general considered the best translation model. Synonymous with the reference translations, the tuning set is also human generated. There are several tuning algorithms used in machine translation systems.

The SMT uses one of the two classes of tuning algorithms: *batch* and *online*. In batch tuning,

Table 2: Example of a reference translation. The input English text describes the code elements in StackOverflow post #13761367

Query	After NLP processing	Reference translation
How to get Latitude Longitude value without using GPS in Android?	latitut longitud valu latitud gps devic network_provid	Location.Location Location.getLongitude LocationManager.getLastKnownLocation Location.getLatitude LocationManager.LocationManager Context.getSystemService LocationListener.onLocationChanged LocationListener.LocationListener LocationManager.isProviderEnabled LocationManager.requestLocationUpdates

the entire tuning set is decoded and new weights are calculated in an iterative process until a convergence. In online tuning, each sentence pair is decoded, then the new weights are updated before the decoding the next sentence. Both algorithms iterate over the tuning set several times before a convergence is reached. Moses has configuration parameter to select the tuning algorithm, with the default being *Minimum Error Rate Training (MERT)* [64]. The other tuning algorithms available with Moses includes *Lattice MERT* [51], *PRO* [36], *MIRA* [83] and *Batch-MIRA*. Schuth [78] and Neubig *et al.* [58] did an exhaustive study of the tuning algorithms used in various machine translation systems.

In our study we use tuning set with Berkeley Aligner, Moses and OpenNMT. We manually curated a tuning set from a random sample of 4870 SO posts. Similar to the test set discussed in Section 3.1.2, we prepared a parallel corpora of English text and code elements from the StackOverflow posts for the tuning set.

3.2 Building the Language Models

In the previous section we described how we create an aligned parallel corpus. In this section we describe the toolkits that we used to build an language model for each SMT technique:

- Berkeley Aligner, to build word based maximum likelihood alignment model.
- *Graph Based API Synthesis (GraSyn)* [61] to perform GraphOrdering using set of code elements as input from Berkeley Aligner alignment model.
- Moses toolkit to perform phrase based machine translation using phrases.

- OpenNMT toolkit to perform neural machine translation with Recurrent Neural Networks (RNN).

3.3 Berkeley Aligner and Maximum Likelihood alignments

Berkeley Aligner [46, 26] is an unsupervised symmetric word aligner. The theory behind the maximum alignment translations was discussed in Section 2. It works on the principle that jointly training two simple asymmetric models will minimize the *Alignment Error Rate(AER)*. This uses a joint training approach whereby the intersection of the two independent directional trained models reduce the overall AER and eliminate the need for garbage collecting rare words. The joint training is done by this agreement, hence it is called *alignment by agreement*. The Berkeley Aligner is written in Java and far easier to configure and execute than Giza++ [65], the only other available IBM maximum likelihood Model implementation. We use the Berkeley Aligner to obtain the maximum likelihood translation and alignment probabilities for our corpus. Building on the Berkeley Aligner, we evaluated the following two algorithms: WordMapK and ContextualExpansion.

3.3.1 Simple WordMapK Algorithm

WordMapK is a naive machine translation approach and serves as a simple baseline in this paper. Using the translation probabilities from the Berkeley Aligner we build a decoding scheme whereby each word of the English query is mapped to the top K code elements. The implementation simply involves ordering the translation probability table in descending order by the probability distribution function. We translate by picking the code elements for each word in the English query and performing a lookup in the table that fetches the top K code elements. The words are stemmed and keywords are extracted before the translation process. This simple algorithm is easy to implement and test but has major shortcomings. The context of the words in the English query is lost in the translation. This simple mapping also provides many irrelevant code elements for consecutive words. For instance, the input "write to a file", after eliminating stop words and extracting keywords would yield "write file". If we pick top K elements for the word "write" will map to both
tt File.write() as well as the incorrect `Socket.write()`.

We measure the precision and recall by varying the value of “K” in the WordMapK algorithm. The parameter “K” defines the number of elements to pick from the translation probability table for each word in the input text. We use 5, 7, 10 and 15 for the “K” value and analyze the outcome.

3.3.2 ContextualExpansion Algorithm

To overcome some of the drawbacks of WordMapK, the ContextualExpansion [61] algorithm identifies the pivotal code elements as those with mappings to the English words in the input. The words are used as a context to derive the pivots. The ContextualExpansion algorithm considers two contexts (word and Code elements). For word context, the next word to be translated must have the highest relative co-occurrence frequency with all the previously translated words in the query. For code context (dependency among elements that were already collected), the next element must have the highest relative co-occurrence frequency with all code elements that were already selected. Continuing our example, the input text "Write to a file" would map to `File.write()` only and not to the incorrect `Socket.write()` because the pivots consider the context "Write" and "file" in the input text. A further expansion is performed based on the mapped code elements. For example, since the element `File.write()` is already selected, there is a strong likelihood that `File.close()` will be need, so it is added to the the expanded mapping. The full details of the approach can be found in Nguyen *et al.* [61].

The ContextualExpansion algorithm had a tuning parameter K similar to WordMapK (top K elements from the translation probability table). We evaluated the performance of this algorithm by varying the K with $(5, 7, 10, 15)$.

3.3.3 Ordering the code into a graph

English is read in a sequential left-to-right manner. In contrast, code is ordered according to a data and control flow graph or tree in the form of *Abstract Syntax Tree (AST)* [37]. To re-order the output of these simple word mapping models, a code only language model is created by mining the graphs of 556 existing android projects using GrouMiner [63]. GrouMiner stores the frequency of occurrence of each code graph during the mining process. The code graph represents the code elements that occur together with control and data dependencies.

To order the code elements from the word map model, we use *Graph Based API Synthesis (GraSyn)* [61] which takes as input a set of code elements and produces a code graph. A beam search is used to generate the code graph that has the highest probability of occurrence in the training database with the given input set of code elements from the mapping models. GraSyn will eliminate nodes which make the overall graph less probable. Since the graph is built by adding one element at a time, GraSyn is not simply a search engine. Indeed 85% of the synthesized graphs do not exist as a whole in the training data, thus, cannot be found by code search. The full details can be found in Nguyen *et al.* [61].

Without the GraphOrdering stage the number of incorrect elements was high. As a result, we

only show the WordMapK and ContextualExpansion results after GraphOrdering. The results before GraphOrdering can be found in the figures and tables in the replication package [66].

3.4 Phrase based translations with Moses

Moses [42] is an implementation of the phrase based machine translation approach the theory of which were discussed in Section 2.4. It also supports hierarchical phrase based machine translation and factored machine translation. Moses has two components: the training pipeline and the decoder. The training pipeline is a collection of utilities which help improve the quality of the bitext. We use these utility scripts for tokenizing, true casing, and cleaning to eliminate noise in our parallel corpora. The cleaning script also removes misaligned translations such as long and empty sentences.

Moses then uses the IBM models (Giza++ [65]) to create word alignments and then extracts phrase based translations through probability estimation. A tuning process add weights to the translation system to improve the translation quality. The tuning process involves finding the weights that maximize translation performance on a small set of parallel corpora, the tuning set. The default tuning algorithm, which we use in this paper, is *Minimum Error Rate Training (MERT)* [64]. MERT is a batch algorithm which requires the whole tuning set to be decoded and model weights updated. The tuning set is then re-decoded with the new weights, the optimization is repeated until the convergence criterion is satisfied. To improve fluency, Moses uses *KenLM*[31] to adjust the language model for output in the target language. This language model is used by the decoder to produce fluent output for the translation. We set the word length of the parallel corpora to 80 words to filter out the misaligned translations.

In our context, the aligned StackOverflow corpus is used to train the model. The posts in the reference translation are translated from the English inputs into a sequence of code elements.

3.5 Neural SMT with OpenNMT

The research into neural machine translation has resulted in many tools including *GroundHog* [28], *lamtram* [57] and *tensorflow-seq2seq*. However, the stability and documentation is often poor. We selected the widely used OpenNMT [39] deep learning framework, which is used in machine translation, speech recognition, and image processing. The theory behind neural sequence to sequence learning models is discussed in Section 2.5. OpenNMT has important features built into its implementation:

- Scripts to support corpus cleaning such as tokenization and word embedding.
- A gated RNN architecture[34, 20].

- Several layers of stacked recurrent neural networks[82].
- Input feeding [50], which entails feeding the previous attention vector back into the input as well as the predicted word.
- Decoding using beam search with support for multiple hypothesis target predictions.

OpenNMT has three implementations for three different implementations namely *LuaTorch*, *PyTorch* and *TensorFlow*. In our study we use the *LuaTorch* version as it is a full-featured, production ready product. Neural machine translation systems are computationally expensive and requires one or more *Graphics Processing Unit (GPU)* to perform the training and tuning of the NMT models. In the absence of GPUs, the training utilizes the *Central Processing Unit (CPU)*. However with CPU's, the training time is several fold slower.

We setup a virtual machine in the Google cloud infrastructure with 4 *Nvidia K80* GPU. The encoder layer of the RNN has 4 layers and the decoder has 8 layers. The size of the neural net is 512 nodes. Each graphics processing unit had about 12 GB of memory and the server had 8 virtual CPU cores with 30 GB of memory. The training was configured for 30 epochs with each epoch spanning $10,000$ iterations totalling approximately 300K iterations. The training program ran for about 4 days in this configuration and completed the 30 epochs which we have configured. We measure the translation performance by changing the beam search width parameter during the decoding process. OpenNMT uses beam search [40] during decoding process. We ran the OpenNMT decoder with two different beam search width parameter. One with the default value of 5 and another decoding by increasing the beam-width to 10 to identify whether the beam-width has an impact on the predictive capability of the decoder. We discuss the results in the section 4.4.

3.6 Ensemble Approach

Ensemble methods is a technique used in statistics and machine learning to combine multiple learning processes to enhance the overall predictive outcome. Dietterich [27] lists three reasons to apply ensemble learning, namely "statistical", "computational" and "representational". Inspired by this ensemble learning, we combined the outputs from different translation systems and measure the performance. For instance we combined the mapping output of the WordMapK translation scheme with NMT and studied how the effective system perform. We discuss the results in the Section 4.5. The commonly used ensemble algorithms are "bagging" [13], "boosting" [29] and "stacking" [89].

In "bagging", different classifiers are trained on randomly selected datasets and then combined. The subset of training data sets are randomly selected with replacement from the entire training data set. The ensemble is then obtained by combining the classifiers in which the class that is most voted

by the classifiers is treated as the ensemble decision. In "boosting" technique, ensembles are created by resampling of the training data and the decision is obtained by majority voting. In "stacking" the ensembles of classifiers are trained using bootstrapped samples of training data and the outputs are trained again in a different classifier, in that the ensembles is formed as stack. We follow a similar approach to "stacking" in that we combine the translated code elements from the different machine translation schemes and then apply GraphOrdering.

3.7 Evaluation Setup

We present the data set and our evaluation details in this section. We use three data sets to evaluate each machine translation system.

- Training data set - a parallel corpus built using the ACE tool with the text and code elements from 236, 919 SO posts.
- Tuning data set - manually curated parallel corpus built with the text and code elements from 4870 SO posts.
- Validation data set - selected reference translations of text and code elements from 240 SO posts.

The training data set is used to build the translation model in the machine translation system. The tuning set is used to optimize the weights and tune the trained model. The validation set is used to evaluate the performance of each machine translation scheme. After the training and tuning process, we use the machine translation system toolkit to translate every input text from the validation set and store the code elements translated by the translation systems in a file corresponding to each input text. We compare this translated code elements with our reference translation in the validation set. For each reference translation from the StackOverflow post, we compute the precision and recall for the code elements in the post. We use the following standard formulas to compute precision and recall.

$$precision = (tp/(tp + fp)) * 100 \tag{14}$$

$$recall = (tp/(tp + fn)) * 100 \tag{15}$$

Table 3 below shows how we compute precision and recall with for a post. In the example translation presented in Table 3, there are 4 true positives and 1 false positive out of the 5 code elements in the candidate translation. Hence the precision value is 80. There are 8 false negatives in

Table 3: Example calculation of Precision and Recall. True positives are highlighted in **bold**. 4 TP and 1 FP out of 5 code elements leads to a precision of 80%. There are 8 FNs for a recall of 33%

Reference translation	**Candidate translation**	**Precision**	**Recall**
Environment.getExternalStorageDirectory File.mkdirs File.File Environment.Environment File.exists File.createNewFile String.String OutputStreamWriter.append OutputStreamWriter.OutputStreamWriter FileOutputStream.FileOutputStream OutputStreamWriter.close FileOutputStream.close	**Environment.Environment** **File.File** **Environment.getExternalStorageDirectory** **FileOutputStream.FileOutputStream** Intent.Intent	80	33

the candidate translation (i.e these code elements are missed by the translation system), hence the recall is 33.33.

We plot the distribution of precision and recall values for the 240 reference translations using a violin plot. A violin plot combines a boxplot and a density plot (shown horizontally). The boxplot is represented as the box in the middle of the distribution. The bottom of the box is the 25th percentile and the top is the 75th. The vertical line is the median. The violin plot on top is precision and recall is on the bottom. Since precision and recall values are skewed for each reference translation, we discuss the median values in the text.

Chapter 4

Results

In this section we present the precision and recall results for each machine translation approach. As an illustrating example, we use the English to code pair presented in Table 2 to contrast the outcomes of the translation systems. The code output for each reference translation can be found in our replication package [66].

4.1 Results for WordMapK

The WordMapK algorithm is a simple maximum likelihood mapping between each English input word and the most probable K code elements. The translation probability table is derived using the Berkeley Aligner. This algorithm is naive and very easy to implement and serves as a baseline to compare the other approaches.

Figure 3 shows the distribution of precision and recall for $K = 5$ and 15 with GraphOrdering. For GraphOrdering we consider K for which we had high precision and recall in the mapping stage. Table 5 summarizes the precision, recall with GraphOrdering. The median precision and recall for $K = 5$ is at 20 and 50 respectively. For $K = 15$ the precision decrease to 8, while recall increase to 66.

Table 4 gives the output of the WordMapK algorithm with $K = 5$ for the input from the test set shown in Table 2. We include the translated code elements from the WordMapK algorithm along with the GraphOrdering stage. The WordMapK algorithm maps the important classes that are used in the reference including LocationManager, Location, LocationListener. We also observe that WordMapK algorithm maps many of the correct methods including Location.getLatitude(), Location.getLongitude(), LocationManager.requestLocationUpdates() which are essential to answer the question, "How to get Latitude Longitude value without using GPS in Android?".

There are many code elements in the translation which are out of context. For instance String,

Table 4: WordMapK output for $K = 5$: Code elements for an English text from the reference translation, *i.e.* true positives, are in **bold**. While WordMapK captures many correct elements it also suggest generic code elements.

WordMapK Output	**GraphOrdering Output**
LocationManager.LocationManager	
Location.Location Location.getLatitude	
Location.getLongitude Context.getSystemService	
LocationListener.LocationListener	**LocationManager.LocationManager**
LocationManager.requestLocationUpdates	**Location.Location Location.getLongitude**
LocationManager.getBestProvider	**Location.getLatitude**
Context.Context String.String	**Context.getSystemService**
LocationManager.removeUpdates	**LocationManager.requestLocationUpdates**
SharedPreferences.SharedPreferences	String.valueOf Context.Context String.String
Toast.makeText Log.Log EditText.getText	LocationManager.removeUpdates
Integer.Integer String.valueOf	
PackageManager.PackageManager	
Address.Address Geocoder.getFromLocation	
Double.Double Build.Build ArrayList.size	

Table 5: Median Precision and Recall of WordMapK with GraphOrdering

K **Value**	**Precision**	**Recall**
5	20	50
15	8	66

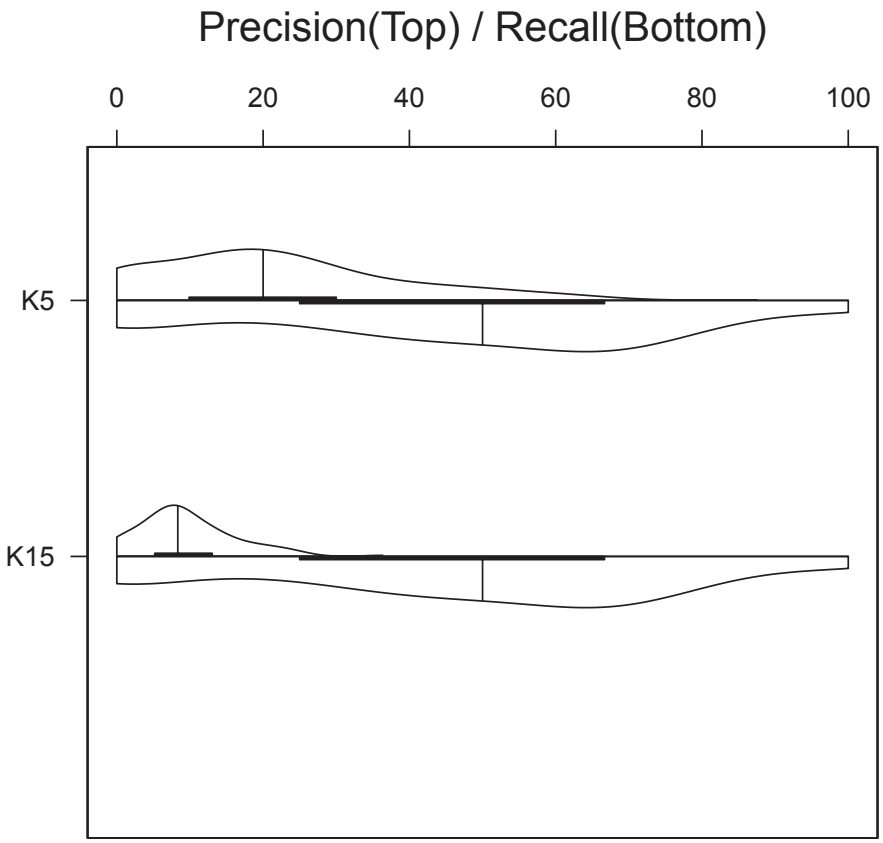


Figure 3: WordMapK with GraphOrdering precision and recall distributions

Table 6: Median Precision and Recall of contextual expansion with GraphOrdering

K **Value**	**Precision**	**Recall**
5	25	40
7	18	42
10	13	50
15	10	50

Integer, String.valueOf(), SharedPreferences, EditText.getText(), Toast.makeText(). We analyze the translation probability table in the Berkeley Aligner and found that the top 5 translations for the keyword "value" that is present in the input text are String, Integer, String.valueOf, SharedPreferences, EditText.getText. We also see several other translations such as Double, ArrayList.size(), Log.Log(), Context, though they were not present in the reference translation data set, these are most common code elements used in solving many software engineering tasks, particularly in Android application development. For instance, considering the translation task, we know that "latitude" and "longitude" are typically stored in a Double variable, hence they are present in our translations even though they are not present in the reference translation.

The GraphOrdering step is essential because it removes the noise inherent in the simplistic WordMapK algorithm. The GraphOrdering eliminates nodes which make the overall set of code elements less probable. Returning to our translation example in Table 4, we see that generic code elements including SharedPreferences, EditText.getText(), Integer are pruned. Despite removing some code elements, others that are commonly associated with the input text are retained even though they are not in the reference translation, including String, LocationManager.removeUpdates(). The output for all the reference translations can be found in our replication package [66].

> WordMapK uses a simple maximum likelihood model to map each English word in the input to K code elements in the output. The elements are then ordered into a graph. The median precision and recall for $K = 5$ is 20 and 50.

4.2 Results for ContextualExpansion

The ContextualExpansion algorithm considers the context of the keywords in the input and maps them to an expanded set of code elements (See Section 3.3.2 for details on the algorithm). Nguyen *et al.* [61] designed this algorithm to overcome the lack of context in the simple likelihood translation table mapping used in WordMapK. The expansion, which is based on earlier translated code elements, takes the software engineering context into account, such as File.open() frequently occurs with File.close().

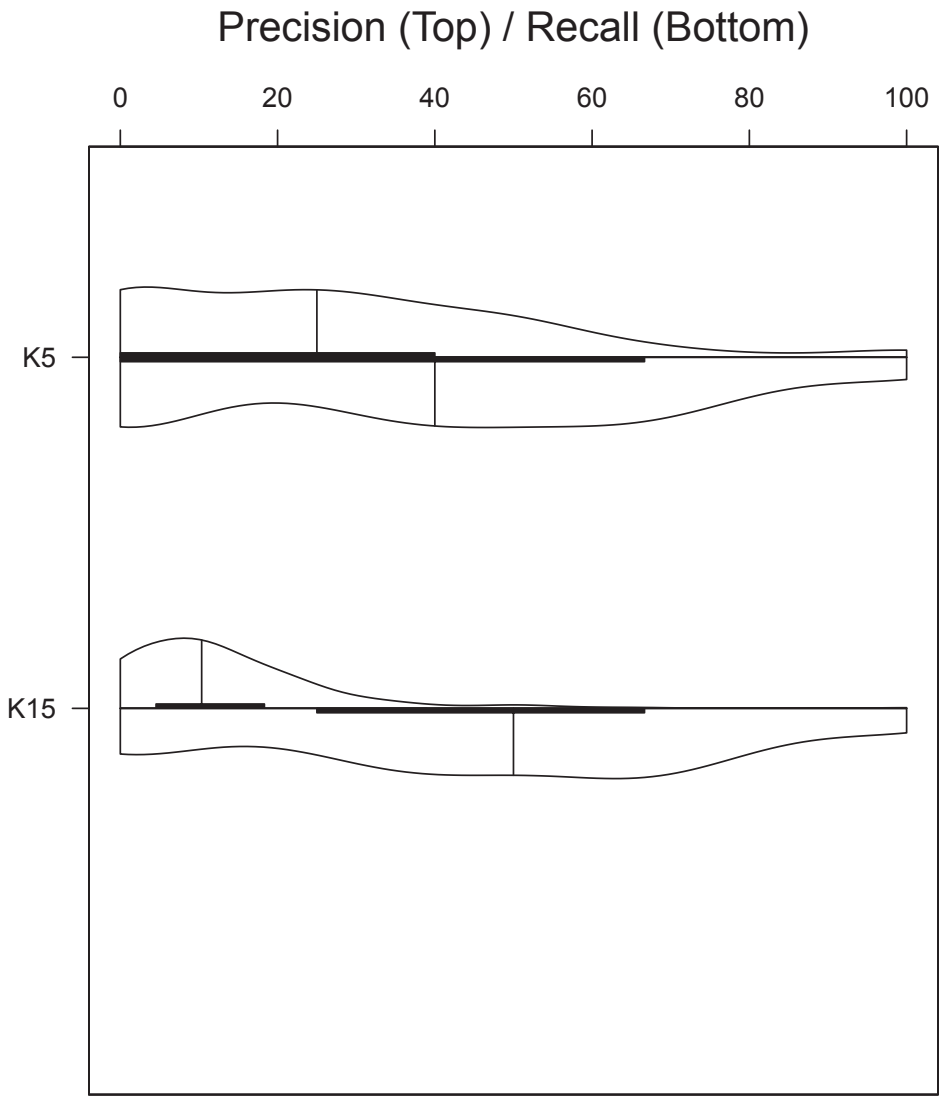


Figure 4: Contextual Expansion with GraphOrdering precision and recall distributions

Table 7: Output of ContextualExpansion (K = 5) Code elements for an English task from the reference translation, *i.e.* true positives, are in **bold**. ContextualExpansion finds code elements that are relevant to the context of the input text.

Contextual Expansion Output	**Contextual Expansion with GraphOrdering**
Context.getSystemService	
Location.Location Location.getLatitude	
Location.getLongitude	
LocationManager.LocationManager	
LocationListener.LocationListener	**LocationManager.LocationManager**
LocationManager.requestLocationUpdates	**Location.Location Location.getLongitude**
LocationManager.getLastKnownLocation	**Location.getLatitude**
LocationListener.onLocationChanged	**LocationManager.getLastKnownLocation**
LocationManager.isProviderEnabled	**LocationManager.isProviderEnabled**
List.List Service.Service	**LocationManager.requestLocationUpdates**
Geocoder.Geocoder Double.Double	LocationManager.removeUpdates
LocationManager.removeUpdates	
Criteria.Criteria	
LocationManager.getBestProvider	
ArrayList.size EditText.getText	

Table 8: Comparing Precision, Recall WordMapK and CE algorithms

	WordMapK with GraphOrdering		CE With GraphOrdering	
K	**Precision**	**Recall**	**Precision**	**Recall**
5	20	50	25	40
15	8	66	10	50

Figure 4 shows the distribution of precision and recall of the ContextualExpansion algorithm with GraphOrdering. Table 6 shows the median precision and recall for $K = 5, 7, 10$, and 15 with GraphOrdering. The median precision is highest for $K = 5$ and recall is highest for $K >= 10$. The best balance is $K = 5$ with precision and recall values of 25 and 40. The ContextualExpansion algorithm outperforms WordMapK for both precision and recall as shown in Table 8.

To put the results in context, from Android GPS translation example, we can see in Table 7 that ContextualExpansion correctly identifies all 10 code elements including the Location.getLatitude(), LocationManager.getLastKnownLocation() method calls. However, it identifies 9 code elements that are not in the reference translation, including the Geocoder, List classes and the LocationManager.isProviderEnabled(), EditText.getText(), ArrayList.size() method calls. Although these code elements are not in the reference translation, they are commonly associated with the English word "Location" as many developers loop through a list of locations or read the text from the "EditText" (i.e textbox) component to obtain a location. On the right side of Table 7 we see the output after GraphOrdering. In this case, 7 of 10 elements are correct. The only incorrect element is the LocationManager.removeUpdates() method, which is commonly used by developers to unsubscribe from location updates from the LocationManager. In contrast to WordMapK, the ContextualExpansion algorithm does not translate out-of-context code elements. For example, the WordMapK algorithm translate 5 incorrect code elements, including String, SharedPreferences, Toast.makeText(), Integer which are generic Android code elements. The incorrect elements translated by ContextualExpansion are those that are useful in the context of location and GPS, including LocationListener.onLocationChanged(), LocationManager.isProviderEnabled(). The output for all the reference translations can be found in our replication package [66].

> ContextualExpansion uses the WordMapK model but incorporates the context of commonly co-occurring code elements. The elements are ordered into a graph. The median precision and recall for $K = 5$ is 25 and 40. ContextualExpansion accounts for context which improves the precision at the expense of recall when compared with WordMapK.

4.3 Results for Phrase Based Machine Translation

The phrase-based machine translation is an extension of word based models with the translation probabilities constructed using phrases (chunks of words). The sequence of code elements are translated from the sequence of English text by mapping the alignment of chunks of text and code elements. The phrase-based machine translation system has implicit reordering capability to generate sequence of words in the target language. Phrase-based translation requires repetition of phrases in the parallel corpora to achieve good translation quality.

Table 9 shows the translated set of code elements for the input text in Table 2. We see that

Table 9: Output of Moses. Code elements for an English task from the reference translation, *i.e.* true positives, are in **bold**.

Reference Translation	Translated code elements
Location.Location Location.getLongitude	
LocationManager.getLastKnownLocation	
Location.getLatitude	**LocationManager.LocationManager**
LocationManager.LocationManager	**Location.Location Location.getLatitude**
Context.getSystemService	Address.getLocality String.String
LocationListener.onLocationChanged	Context.Context Address.getAddressLine
LocationListener.LocationListener	Geocoder.Geocoder
LocationManager.isProviderEnabled	
LocationManager.requestLocationUpdates	

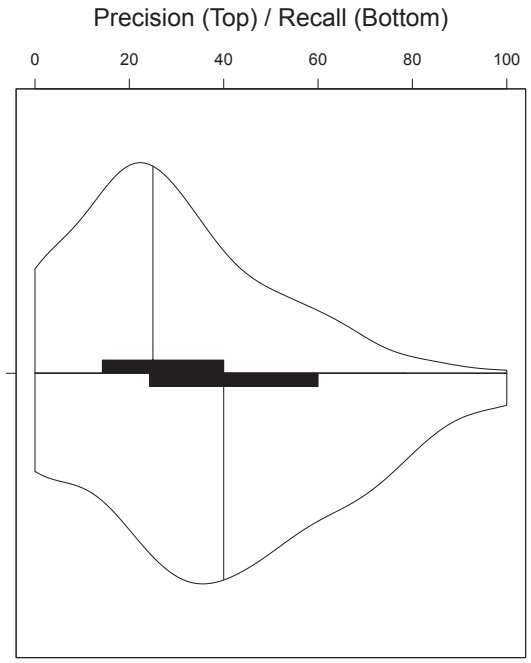


Figure 5: Moses precision and recall distribution

Table 10: Comparison of Moses with CE and WordMapK algorithms. The true positives are highlighted in **bold**

Input text	**WordMapK (K = 5) with GraphOrdering**	**ContextualExpansion (K = 5) with GraphOrdering**	**Moses Output**
play media file continuously	**MediaPlayer.MediaPlayer** **MediaPlayer.setDataSource** **MediaPlayer.start** **MediaPlayer.stop** View.View MotionEvent.MotionEvent MediaPlayer.release MotionEvent.get ActionView.setOnClickListener	**MediaPlayer.MediaPlayer** **MediaPlayer.start** Log.d Intent.putExtra Toast.Toast Toast.makeText	**MediaPlayer.MediaPlayer** Log.d View.View
Sending email on android via default email application	**Intent.Intent** **Intent.putExtra** Context.Context Intent.addCategory Context.startService Intent.setType Intent.createChooser	**Uri.Uri** **Uri.parse** **Activity.startActivity** **Intent.putExtra** Intent.setType Intent.createChooser	**Intent.Intent** **Intent.putExtra** Intent.setType Intent.createChooser
Android Crop Center of Bitmap	**Bitmap.Bitmap** **Bitmap.createBitmap** **Bitmap.getWidth** View.View Intent.putExtra Intent.setDataAndType Canvas.drawBitmap View.getHeight Display.getWidth Intent.setType TextView.TextView TextView.setGravity Display.Display Intent.Intent	**Bitmap.createBitmap** **Bitmap.getWidth** **Bitmap.Bitmap** **Bitmap.getHeight** Canvas.Canvas View.getHeight Matrix.Matrix Matrix.postScale Canvas.drawBitmap View.getWidth	**Bitmap.createBitmap** **Bitmap.Bitmap** **Bitmap.getWidth** Bitmap.getConfig Canvas.Canvas String.String RelativeLayout.RelativeLayout Canvas.drawBitmap Bitmap.copy Bitmap.createScaledBitmap Bitmap.Config

Moses translates less code elements when compared to the ContextualExpansion and WordMapK algorithm for the same input text. Moses is able to translate only 3 code elements out of the 10 code elements from the reference translation. We see that Moses is able to predict code elements such as Geocoder class and Address.getAddressLine(), Address.getLocality() methods which are relevant to the "Location" keyword.

Figure 5 shows the results of precision and recall at the output of Moses. We analyzed the sequence of code elements translated by Moses for each of the input text from our validation data set and observed that length of the translated code elements is less than that of naive WordMapK and ContextualExpansion algorithms with graph ordering. The median precision and recall is 25 and 40 respectively for the entire set of reference translations.

We compare the output from WordMapK and the ContextualExpansion algorithms with Moses. We observe that although the overall precision and recall is similar for Mosses and ContextualExpansion, the sequence of translated code elements for Mosses is much more limited. As a result, in Table 10 we present three additional reference translations to help compare phrase based machine translation with ContextualExpansion and WordMapK algorithm. The output for all the reference translations can be found in our replication package [66].

The first example in the Table 10, the question is how to "play media files continuously?" We find that WordMapK and ContextualExpansion algorithms are able to translate relevant code elements including MediaPlayer.setDataSource(), MediaPlayer.start(), MediaPlayer.stop() along with the class MediaPlayer whereas Moses is able to translate only the class without any methods. Moses is unable to find likely code elements which coexists with MediaPlayer class. In the second example, which describes sending email with the default application, we find that Moses is able to translate two code elements namely the Intent class and the Intent.putExtra() method. These are very common code elements for Android. ContextualExpansion provides a more accurate translation adding the Uri.parse() method. In the third example, which describes cropping a bitmap, all three approaches find the Bitmap.getWidth(), Bitmap.createBitmap() methods coexisting with the Bitmap. ContextualExpansion is able to also correctly identify Bitmap.getHeight() which is commonly associated used when both width and height are required.

We hypothesize that Moses is able to translate the coexisting code elements and relies upon the occurrence frequency of the co-occurring code elements (i.e phrases) in the parallel corpora to perform the translations. From Table 10 we see that Moses is able to translate code elements from English text with similar precision to that of ContextualExpansion with GraphOrdering. In terms of the median precision on the overall validation data set, Moses performs better than the naive WordMapK algorithm and is competitive with the ContextualExpansion algorithm with GraphOrdering.

Table 11: Output of OpenNMT. Code elements for an English text from the reference translation, *i.e.* true positives, are in **bold**.

Reference Translation code elements	Translated code elements
Location.Location Location.getLongitude LocationManager.getLastKnownLocation Location.getLatitude LocationManager.LocationManager Context.getSystemService LocationListener.onLocationChanged LocationListener.LocationListener LocationManager.isProviderEnabled LocationManager.requestLocationUpdates	**Location.Location**

Table 12: Median precision and recall in OpenNMT

Beam-width	Precision	Recall
5	80	20
10	100	20

Phrase-based translation translates sequences of words. As a result, English phrases are mapped to code phrases. The median precision and recall is 25 and 40. Phrase-based translation and ContextualExpansion produces similar translated outputs.

4.4 Results for Neural Machine Translation

OpenNMT uses artificial neural networks and deep learning to perform the sequence to sequence translation. Neural machine translation is the current state-of-the-art for machine translation and is used by Google, Microsoft, and Yandex [12]. NMT has shown significant improvements in the translation quality and is well suited for sequence to sequence translation [10]. The target language sequence prediction is based on complete source sentence and the target sentence that has been produced already. We expect neural machine translation to translate long sequence of code elements with high precision.

OpenNMT uses beam search [40] during translation process. Beam search algorithm uses a width parameter to limit the search space. We evaluated OpenNMT toolkit with two different beam search

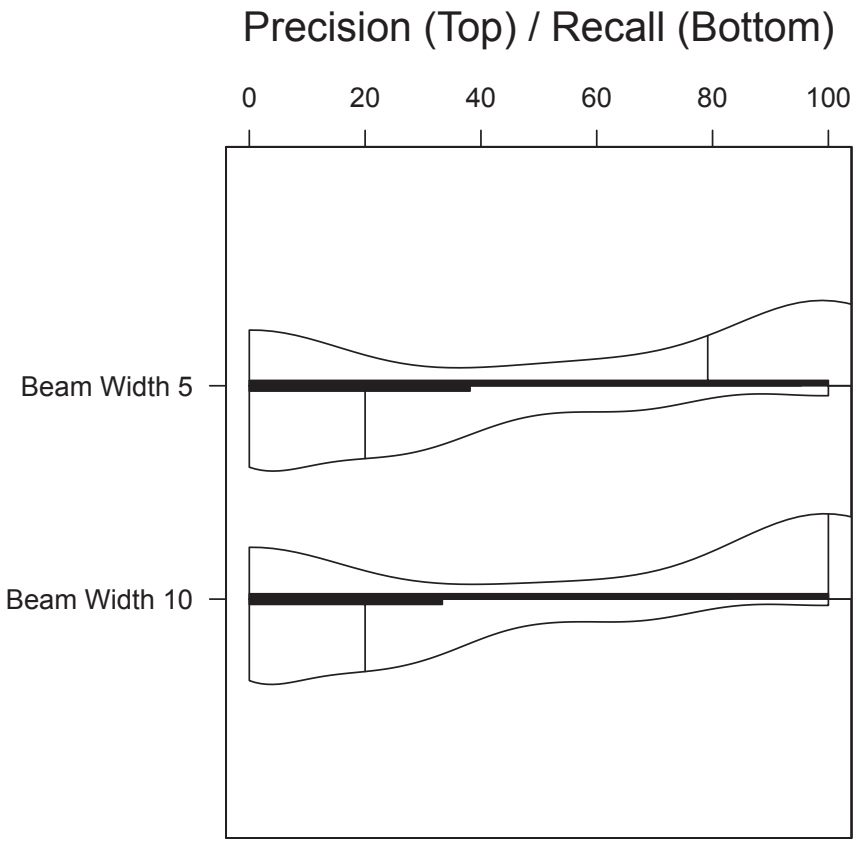


Figure 6: Neural Machine Translation precision recall distribution

width parameter. One with the default value of 5 and another by increasing the beam-width to 10 to identify whether the beam-width has an impact on the predictive capability of the translator.

Table 12 shows the median precision and recall for OpenNMT for two beam-width configuration and Figure 6 shows the corresponding distribution. We observe that a median precision and recall of 80 and 20 for beam width of 5. When we increase the beam width to 10 the recall remains the same but the precision increases to 100. We manually analyzed the translated output between beam-width 5 and 10. We found that 95 percent of the input texts the change beam-width parameter translated the same code elements. With a beam-width of 10 there were a few additional correct code elements

Table 11 shows the GPS "location reference" translation example. OpenNMT is only able to translate the class Location. OpenNMT is precise, however it does not translate enough code elements (i.e suffers recall) for the given input text. The translation of few code elements is present in other reference translation. In the "media player" example only MediaPlayer.setOnCompletionListener() is translated. In the "sending email" example, only the Intent class is translated. In the "crop Bitmap" example, only the Bitmap class is correctly translated. We analyzed the output for all the reference translations and found that for most only constructor of the classes were the only translations. OpenNMT translates relatively few methods for input texts compared to the class constructors.

> Neural networks translate sequences of English to code sequences. The median precision and recall are 100 and 20. Unfortunately, the output is usually limited to a small number of class constructors and typically does not provide interesting API usages.

4.5 Results for Ensemble Translation

Ensemble learning is used in machine learning to combine multiple learners to enhance the overall predictive outcome [27]. Section 3.6 provides the the theoretical background on ensemble learning. Inspired by ensemble learning, we combine neural machine translation which had high precision with the WordMapK and GraphOrdering ($K = 15$) algorithm which had the highest recall. Figure 7 presents the ensemble results. To aid the comparative analysis we also include the precision and recall distribution in the GraphOrdering step of the WordMapK ($K = 15$) algorithm. The median precision and recall of this ensemble of neural machine translation and WordMapK ($K = 15$) is 20 and 50 respectively, which equals the median precision and recall obtained in WordMapK ($K = 15$) scheme with GraphOrdering. We conclude that the neural machine translation did not predict substantially different code elements than the naive WordMapK algorithm. The discussion section we further discuss potential reasons.

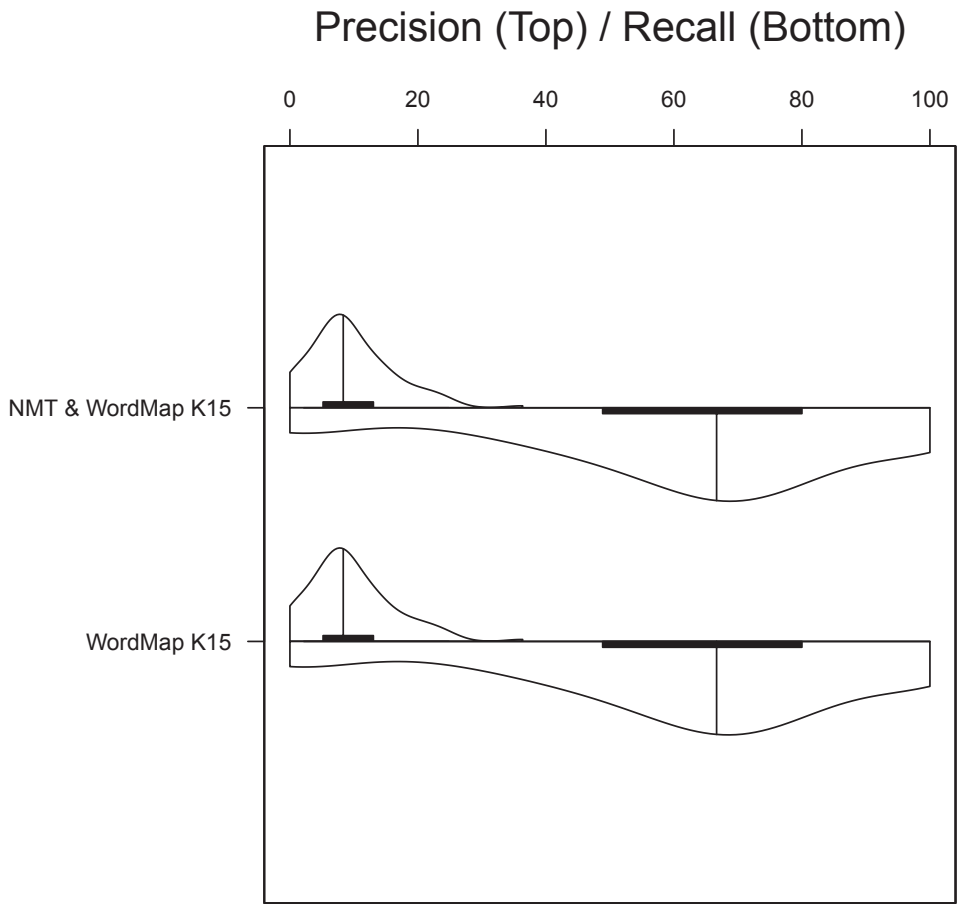


Figure 7: Distribution of precision and recall with code elements from OpenNMT and WordMapK(k = 15) with graph ordering

Chapter 5

Discussion and Related Work

In this thesis we have compared four existing machine translation approaches: WordMapK, ContextualExpansion, phrase-based, and neural network translation. Table 13 shows the resulting median precision and recall across the set of reference translations. The WordMapK algorithm which is a simple maximum likelihood model predicts the largest number of code elements for each input word leading to high recall. It is however, quite imprecise predicting irrelevant code elements that lack the context of the input. ContextualExpansion which build on WordMapK by considering the context has a higher precision with a compromise in recall. ContextualExpansion strikes a balance between precision and recall. Phrase-based translation translates short sequences of English to Code. The precision and recall are similar to that ContextualExpansion perhaps because both consider the context of each word and code element. Neural machine translation has the highest precision, however, it conservatively suggests class constructors instead of meaningful sequences of code elements.

5.1 Overlap in Predicted Elements

Precision and recall provide a comparison between techniques based on the elements that each technique correctly predicts. We are interested in determining how similar the code elements

Table 13: Summary of results

Method	**Precision**	**Recall**
WordMapK ($K = 5$) with GraphOrdering	20	50
ContextualExpansion ($K = 5$) with GraphOrdering	25	40
Moses	25	40
OpenNMT	100	20

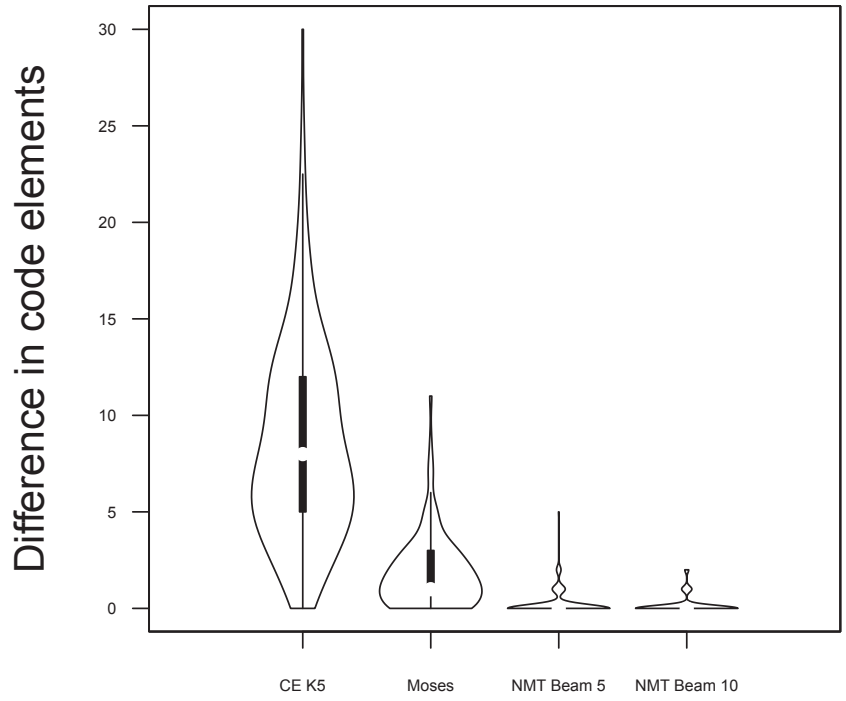


Figure 8: Set difference of code elements from CE, Moses and OpenNMT against WordMapK

Table 14: Computational Cost: Hardware and Maximum Processing Time

Toolkit	**Hardware**	**Time**
Berkeley Aligner	1 CPU, 2GB RAM	10 minutes
GraphOrdering	8 CPU Cores, 24GB of RAM	4 hours
Moses	8 CPU, 24GB of RAM	16 hours
OpenNMT	4 GPUs, 48GB GPU Memory	4 days(GPU)

are among predictors. We calculated the elements that are unique to each technique related to WordMapK. We choose WordMapK because it suggested the most code elements. Mathematically this is the complement or set difference with WordMapK: $ContextualExpansion \setminus WordMapK$, $Phrase \setminus WordMapK$, and $NMT \setminus WordMapK$. We plot the number of code elements in the complement for each technique in Figure 8.

In the figure, we see that the ContextualExpansion suggests a median of 8 code elements that are not predicted by WordMapK. This suggests that ContextualExpansion not only provides contextually relevant code elements, it also suggest additional elements that are missed by simple word mapping.

In contrast, the phrase-based translation and NMT prediction suggest few additional elements relative to WordMapK. The median complement is 1 and 0 for phrase-based and NMT, respectively.

The prediction from the neural and phrase based machine translation techniques are nearly subsets of the WordMapK prediction. This subset finding makes it clear why we cannot combine the phrase based or neural machine translation translated set of code elements with the code elements obtained from WordMapK technique.

5.2 Computational Costs

The hardware used in our experiments and time taken to execute each machine translation technique are shown in Table 14. The word mapping Berkeley Aligner is least expensive in terms of resource requirements. It can be run on a standard desktop computer or laptop which has moderate memory and number of CPU cores. In our experimental setup, the Berkeley Aligner took about 10 minutes to align the more than 200k posts in the StackOverflow corpus. The Berkeley Aligner is the limiting factor for WordMapK and ContextualExpansion does not add significant additional time.

The GraphOrdering step require requires a large graph database that includes mining 543 Android projects from GitHub. The graph database when loaded from disk expands to 12 GB of main memory. Hence the GraphOrdering steps requires a strong server. We used a server with 8 cores and 24 GB of RAM. The GraphOrdering phrase is dependent on the size of K, with a large K = 15 this phase took

4hrs.

The phrase-based Moses toolkit requires a strong server [56]. The training time is proportional to the number of cores. Running on an 8 CPU machine with 24GB of RAM the training took 16 hours.

Neural networks require substantial processing power. Our first attempt was to run OpenNMT on the 8 CPU machine with 24GB of ram. After running the training process for two weeks, we estimated that it would take at least 30 days to complete.

We switched to a Google Cloud GPUs. We set up a virtual machine with four NVidia K80 GPUs. Each GPU had 12GB of GPU memory. The server had 6 virtual CPUs with 30 GB of RAM. With this configuration, the time taken for one iteration is less than 2 seconds and whole training completed in less than 4 days with the total cost of 250 Canadian dollars. Though we included four GPU's in the server configuration, OpenNMT did not utilize the memory of all the GPU's. In total, we had 48 GB of GPU memory of which the OpenNMT consumed 12 GB during training. Each *NVidia Kepler 80* GPU costs about $6,000$ Canadian dollars. For a similar configuration in an in-house setup, we would have ended up spending nearly $20,000$ Canadian dollars. Though cloud infrastructure provides an alternative option for setting up an inexpensive computational environment, it is not suitable for recurrent executions. In summary neural machine translation toolkits requires an expensive investment in infrastructure for early feedback and recurrent experiments.

5.3 Threats to validity

The number of iterations can be adjusted for the Berkeley Aligner. We adjusted this parameter for WordMapK and ContextualExpansion. Although more iteration usually improves the translation quality, we found the best results with 2 iterations. Indeed, the precision and recall values start decreasing for 5 to 20. However for iterations $20, 50, 100, 200$ the median precision and recall remain constant but they are not better than two iterations. We reported two iterations for all our experiments in this study. The results for iteration settings can be found in the replication package [66].

With phrase-based translation, imbalanced sequences can affect the Moses toolkit. An imbalance occurs when the number of English words is substantially more than the number of code elements in for a a post in the bitext. Furthermore, Moses is unable to handle sentences longer than 80 words in source and target language. Hence we clean the corpora before the training process. This reduced our training data set for Moses. The initial size of the corpus is 331K lines of bitext. We ran the cleaning script provided by Moses. Lines with imbalance were pruned resulting in a bitext of size 224K. This bitext is smaller than the original corpus used by the other machine translation techniques.

Machine translation is typically assessed with metrics such as BLEU [67] and METEOR [11].

These metrics are designed for natural languages by comparing that the sequence of tokens in the translation, *i.e.* n-grams, with the reference translation or manual translation. Unlike natural languages that are read sequentially, *e.g.,* left-to-right, code is represented as a graph. It is unclear how BLEU can be modified to work on graphs. As a result, we resort to using the precision and recall of the translated elements. We also use the set differences among technique as a further comparison.

5.4 Related Work

5.4.1 Mining API Rules

In modern software development, frameworks and API's form a major backbone. Hence identifying the API usage pattern is an important outcome in the source code mining. Michail [53, 54] developed *CodeWeb* which uses *association* and *generalized association rules* to mine frequently reused API methods and classes in a library. Xie *et al.* [90] developed *MAPO (Mining API usages from Open Source Repositories)* which mines the frequently used API usages and present them for developers to inspect.

Li and Zhou [45] used the *frequent item set mining* to extract implicit programming rules from large corpora of industrial projects. The mined rules are used to detect violations in the source code. Acharya *et al.* [6] developed a framework to capture frequent partial orders of API calls among the related API usages directly from the API client code. Uddin *et al.* [85] applied machine learning techniques to model the coherent temporal changes in the API usages of the client program. Liu *et al.* [49] developed a model checker to automatically extract software library usage patterns that can be elicited as temporal properties. Mileva *et al.* [55] measured the success of an API through its popularity. Wang *et al.* [86] developed *Usage Pattern Miner (UP-Miner)* which mined the API usages with measure of succinctness and coverage. Linares-Vasquez *et al.* [47] empirically studied the energy greedy API usage patterns in Android applications using a hardware power monitor. Azad *et al.* [9] used association mining rules to find commonly co-occurring API elements on StackOverflow and suggest autocompletions to developers. Unlike these works, we translate from English input texts to code.

5.4.2 Code Search and Recommendation

Atkins *et al.* [8] developed *Version-Editor* which integrates the past versions of the source code using the history from the version control systems to assist developers while making changes in the software by using the version histories. Cubranic *et al.* [24] developed *HipiKat* an Eclipse plugin which is an implicit group memory from project archives and recommend artifacts relevant to the tasks that is

performed by the developer. Yunwen Ye [91] developed *Code Broker* an intelligent software agent that monitors the current programming activity and automatically delivers the task-relevant Java library components that are not known to a programmer. It will also find an example program that uses the programmers chosen component. Ying *et al.* [92] developed a source code recommender which leverages the information from the version control and suggests the additional set of files that has potential changes based on the current set of files that are modified by the developer. Holmes *et al.* [35] developed *Strathcona* an *Example Recommendation Tool* which leverages the structural characteristics of both the past projects and the current developer context to recommend relevant examples. Sahavechaphan *et al.* [76] developed an Eclipse plugin named *XSnippet* a code assistant system that facilitates developers with object instantiation task at hand by accepting a query to find code snippets from a repository.

Lemos *et al.* [44] developed a tool named *CodeGenie* which employs a test-driven code search on large code repositories and presents the resulting code from the search adapted to use for the developers project. Reiss *et al.* [71] built a Java based Web interface to accept as input the specifications as precisely as the class, method signatures, test cases and security constraints and performs a semantics based code search. Little and Miller [48] implemented *Keyword Programming* for Java language as an Eclipse plugin which accurately constructs 90 percentage of original expressions from keywords provided as input queries. Thummalapenta *et al.* [84] implemented a web based code search application which accepts the queries in the form *Source object ==> target object* and interacts with a code search engine to refine the results and present the relevant code snippets. Alnusair *et al.* [7] presents a code reuse tool which uses ontological representation of the source code without the need for explicit source code repository. Their algorithm analyzes the library API and builds the ontological information on the fly by traversing the code graph. They accept the queries of form *Source object ==> target object* from the developers and presents a code snippet by considering the context of the current code. Instead of searching for and recommending common code snippets, we use statistical machine translation to create potentially unseen code snippets based on an English input text.

5.4.3 Natural Language Processing and Machine Translation

Use of NLP and machine translation systems has recently become more common in the field of software engineering. Hindle *et al.*'s [33] research showed that the source code used in programming exhibits a repetitive nature synonymous with the natural languages. Hindle *et al.* studied the naturalness of software and used n-gram language model to build a code completion engine for Java. They presented the repetitive nature of the source code mathematically by using the theory of statistical language modelling.

Karaivanov*et al.* [38] used phrase based machine translation to transform source code from C-sharp to Java. They trained on a large parallel corpora of source code elements from C-sharp and Java languages. They found that applying phrase based translation as is to the source code transformation does not work and produce grammatically incorrect translations because of the inherent structure enforced by the grammar of the programming languages. Hence they extended the phrase based translation system to accommodate the grammar of the languages.

SWIM [69] uses a statistical translation system (word based model) to perform a mapping from the clickthrough data obtained with the Bing search results and then uses a *Word2Vector* to reorder the keywords to code elements. SWIM is based on the C-sharp API for the training and validation.

DeepAPI [30] uses deep learning approach with sequence-to-sequence based *Recurrent Neural Network (RNN)*. DeepAPI neural language model is built from the Java projects from GitHub by performing a mapping between the first line of the JavaDoc comment and the code elements used in the body of the corresponding method.

Nguyen *et al.* [61, 60] developed T2API to address the shortcomings in the sequence to sequence translation models with *GraSyn* which generates a code graph using graph based language model of the code. T2API process the discussions in the StackOverflow platform to prepare a parallel corpus of English text and source code. It then use a statistical word based model to align the English text and code elements, which is transformed into a code graph using *GraSyn*.

The previous research into translation of English input to code output have evaluated only a single translation approach. In contrast, we use a common dataset and evaluate the historically most popular natural language translation approaches to assess their appropriateness for use in software engineering.

5.4.4 Comparison with Other Statistical Approaches in Software Engineering

In this section we compare our study with similar research that has used statistical approaches in the fields of code search, synthesis and migration. In our study we used unmodified implementations of neural machine translation and phrase based machine translation systems to translate English text to API sequence calls. We find that the precision and recall with the existing machine translation systems are quite low compared to that of past research which use *Information Retrieval (IR)* approaches. We analyze the performance of similar research and present justifications to make software engineering specific changes to the machine translation and information retrieval systems.

We will first analyze research that uses information retrieval techniques perform code search and suggestion. PARSWEB [84] uses *Google Code Search Engine (GCSE)* results for the query of type "Source -> Destination" where "Source" and "Destination" represent object types. The objective

of the query is to obtain the destination object type given the source object type. PARSEWEB uses an *Abstract Syntax Tree (AST)* parser and builds *Directed Acyclic Graph (DAG)* from the code samples obtained from the GCSE. It then extracts the *Method Invocation Sequence(MIS)* from the code samples to rank and list the possible MIS from the source type to the destination type. They show that using code search results from GCSE without processing with AST and DAG cannot solve the problem of finding relevant method invocation sequences.

Ponzanelli *et al.* [68] developed *PROMPTER*, a tool that automatically retrieves relevant discussions from StackOverflow. The tool considers current context of code in the IDE and transforms into a query, which is then sent to a custom code search engine to retrieve relevant StackOverflow discussions and rank them. It applies different features including *textual similarity, code similarity, API types similarity* to rank the resulting StackOverflow discussions. The evaluation was conducted with programmers of varying skills and measured relevance of StackOverflow discussions. They reported that developers using *PROMPTER* were able to complete a software engineering task about 68% of time.

In the context of statistical modeleling in software engineering, Hindle *et al.* [33] studied the repetitiveness of software by using n-gram models. They developed a code suggestion tool based on this model which proves to improve the existing code suggestion component of Eclipse IDE. They relied upon the lexical information in the source code. However lexical tokens cannot capture the abstract semantics of the source code. *Semantic LAnguage Model for source Code (SLAMC)* by Nguyen *et al.* [62] improved the code suggestion by using semantic annotations which they call sememes instead of using lexemes. SLAMC provided improvement in the code suggestion compared to the n-gram models used by Hindle *et al.*. They empirically evaluated in different settings including a cache of recently used variable names, data dependencies among the n-gram tokens and reported improvements over the standard n-gram model based code suggestion. It should be noted that the n-gram model works well with natural languages and assumes a left-to-right sequence of tokens. In contrast, source code is typically represented in the form of a graph. Compilers and humans do not process source code sequentially similar to that of natural languages[22, 23, 75, 17]. Nguyen *et al.* [63, 59] have shown that the *Graph Based Object Usage Model (groum)* can be used to suggest source code elements at a higher level of abstraction than an n-gram model. So the statistical modelling cannot be used for source code suggestion without augmenting them with software engineering specific information.

In the context of machine translation, the work of Karaivanov *et al.* [38] of using phrase based machine translation system (PBMT) to translate source code from C-sharp to Java programming language. In their study they used the phrase based machine translation without any modification. However it was found that the resulting translations to Java code had syntactic and compilation

issues. Hence they augmented the phrase based machine translation system with the prefix grammar of the languages (C-sharp and Java) as part of the translation. The prefix grammar ensures that the previously translated code elements are parseable and discard non-parseable code elements. They report BLEU [67] as well as parse rate as the measure of quality, with the claim that BLEU score is inappropriate for the programming languages. It was reported that unmodified PBMT system translates code that fails to parse in more than 60% of test cases, whereas the modified PBMT system using prefix grammar failed to parse only in less than 1% percent of test cases. So this study reveals that we cannot use statistical machine translation as is without considering the software engineering context (i.e programming language).

SWIM [69] uses word based model [16] to map user queries to the API elements. They use the clickthrough data from the *Bing* search engine and extract API elements in the resulting web pages. This information was used in the mapping model to translate user queries to API elements. They separately mine 25000 open source Github projects to extract structured call sequences for each object type in the C-sharp framework. The statistical word alignment system used to map the queries to API elements does not include order of the API elements, control, data flow information and variable names. Hence they designed an API synthesizer which takes as input a set of API elements and combine them to a valid code snippet. SWIM reported the percentage of relevant API elements in top 5 and top 10 synthesized code snippets. They report that about 65% and 54% of synthesized snippets from top 5 and top 10 respectively are relevant.

DeepAPI [30] translates from English text to API sequences. DeepAPI does not synthesize code. They extended the neural translation model with *Inverse Document Frequency (IDF)* based weights for the API elements. For example, frequently appearing API elements including Logger.Log, System.out.println are given lower weights and less common API elements will be given higher weights. A modified cost function is applied to the RNN Encoder-Decoder model to enhance the translation. The training parallel corpora used in DeepAPI is prepared from using the first line of JavaDoc comment and the list of API elements found in the method body. This parallel corpora is relatively noise free when compared to English text and code elements extracted from StackOverflow. DeepAPI uses BLEU score as a measure of the translation quality and achieves a better BLEU score compared to that of translation of natural languages.

T2API[61] extended the word based alignment model[15, 16] with context information of API elements and English text. The context information is derived from computing the co-occurring English texts and API elements in the parallel corpora. T2API uses Graph based language model of the source code to search and synthesize code graph using a set of API elements as input. We have shown in our study that the median precision and recall in the contextual expansion with GraphOrdering performs only marginally better than the baseline approach WordMapK and matches

to that of the unmodified PBMT system. This could be partially because of the inherent noise present in the extracted parallel corpora from StackOverflow.

Campbell *et al.* [18] developed *NLP2Code* an IDE based content assist solution which allow developers to integrate with source code from StackOverflow. The plugin offers a content assist system loaded with natural language tasks extracted from StackOverflow posts that are tagged as "Java". They use Stanford NLP toolkit [52] to find relation of words in the sentences that are obtained from the StackOverflow. The query selected by the user is then sent to the custom Google search engine on StackOverflow and ranks the resulting code snippets. They claim that *NLP2Code* outperforms T2API with a set of evaluation queries.

In T2API, DeepAPI, SWIM and Karaivanov *et al.* [38] we observe that the statistical machine translation models designed for natural languages does not work well for source code generation or translation without augmenting them with software engineering specific information. Our results suggests that Contextual Expansion with GraphOrdering used in T2API is able to synthesize many unseen API elements compared to the phrase based machine translation system. The contextual expansion considers code and text context into consideration during translation, whereas the phrase based machine translation system works on aligned sequence (phrase) of text and code elements.

The systems using information retrieval techniques tend to outperform the statistical systems as they focus more on code search and code retrieval from existing repositories and examples. However they do not focus on synthesizing new code which could explain the fact that they tend to outperform the statistical systems for similar set of natural language queries.

Chapter 6

Tools: T2API and CompareSMT

T2API [61, 60] is a context-sensitive statistical machine translation approach that takes a description of a software engineering task in English and synthesizes an API usage template. T2API research involves several components including word alignment with Berkeley Aligner, building contextual mapping model using translation probability table from Berkeley Aligner, computing co-occurrence frequency and synthesizing API usage from a trained Graph database. The development of an integrated pipeline and a user interface to test translation is discussed in this chapter. We implement a web portal as well as *Representational State Transfer (REST)* APIs to deliver the components in the T2API system for various client uses.[1] We then extend the same portal to include an interface to compare the translation outcomes of multiple machine translation systems for the same English query (CompareSMT).[2] The web portal includes three interfaces to support following usecases:

- To accept an English text description of the software engineering task and present an API usage template with code graph. This interface uses the ContextualExpansion algorithm to translate English text to source code.
- To display stage wise execution of translation of English text to source code with ContextualExpansion. This allows experimenters to determine the productivity of each individual stage.
- Accept an English text and display source code translations outputs from WordMapK, ContextualExpansion, Moses, and OpenNMT in a single view to compare their performance.

[1] T2API: `https://users.encs.concordia.ca/~pcr/ToolT2API.html`
[2] CompareSMT: `https://users.encs.concordia.ca/~pcr/CompareSMT.html`

6.1 Components

T2API is comprised of multiple independent components. Some components works offline and produce an output which can be used later for the integrated pipeline. The components that work offline include

- Preparation of parallel corpus by extracting English text and source code elements from StackOverflow.
- Training of parallel corpus with machine translation systems including Berkeley Aligner(Word-based model), Moses(Phrase based), OpenNMT (Neural machine translation).
- Preparation of graph database by mining open source Java Android projects from github with GrouMiner [63].

The online components are implemented as REST APIs that accept English text as input and invokes the decoder (i.e translator) part of the machine translation system. For instance, if a user invokes the REST interface to decode an English text with phrase-based machine translation system (i.e Moses), the *moses decoder* will be executed. The online components provide the following services in the form of REST interfaces including

- A service to perform NLP techniques including stemming, stop word removal, and keyword extraction on the English input.
- A service for ContextualExpansion that computes mapped code elements for the input.
- A service that produces API code template for a given set of mapped code elements obtained with ContextualExpansion.
- A service to retrieve the API usage graph images.
- A service to invoke the decoder for word-based(WordMapK), phrase-based(Moses), and neural network models (OpenNMT).

6.2 Architecture

The T2API components were developed in isolation as part of the earlier research. The components were implemented to perform input/output operation through files. We designed and implemented the pipeline to transfer data from one component to another using files. We create REST end points to execute the components in isolation. The web client combines and presents the outcome from each stage. We create a web interface to accept an English text input describing a software engineering

task and present the top ranked API template graph. The T2API web portal is made up with the following frameworks and components.

- Jersey[4] is a framework to develop the REST API.
- AngularJS[2], HTML, and CSS to handle the user interactions on the web client.
- Apache Tomcat[3] to host the web application.

The figure 9 shows the architecture of T2API console. The ContextualExpansion, GrouMiner and GraphOrdering components were developed in Java programming language as part of the earlier research of T2API. Hence we decide to use Apache Tomcat framework to host the Java web application and Jersey library to build REST interfaces on top of existing components and the new pipeline connecting those components. The web application use the translation probability table from Berkeley Aligner to translate English text to API elements using WordMapK and ContextualExpansion algorithms. The web application loads the trained graph database into the main memory during the initialisation. For the phrase based machine translation decoding it loads the *Minimum Error Rate Training (MERT)* model file and invokes the Moses decoder. Similarly for neural machine translation system, the decoder component of the OpenNMT toolkit is invoked with trained neural translation model file.

6.3 Server Components

The server comprise of REST interface implementations for various URLS. The REST interfaces are designed to accept and respond with *Javascript Object Notation (JSON)* data interchange format. For every invocation of the REST URL, the server creates a random *Globally Unique Identifier (GUID)* and creates a directory in the server filesystem to isolate the execution of different components in simulataneous requests threads.

6.3.1 Stemming

Stemming is the process of reducing inflected words to the root form. For example words such as "fishing", "fished" and "'fisher" would be represented in the stemmed form as "fish". The raw query in English has to be converted to a common base form. This will reduce inflections and remove derivational affixes. We use the English Snowball Stemmer [5] which is a *Java ARchive (JAR)*, which takes as input a file containing raw English text in each line and produces an output file contained stemmed words corresponding to each line. We execute the stemmer as a separate Java process from within the T2API server. The raw English text received from JSON request is written to a file and

Figure 9: T2API Web Console Architecture

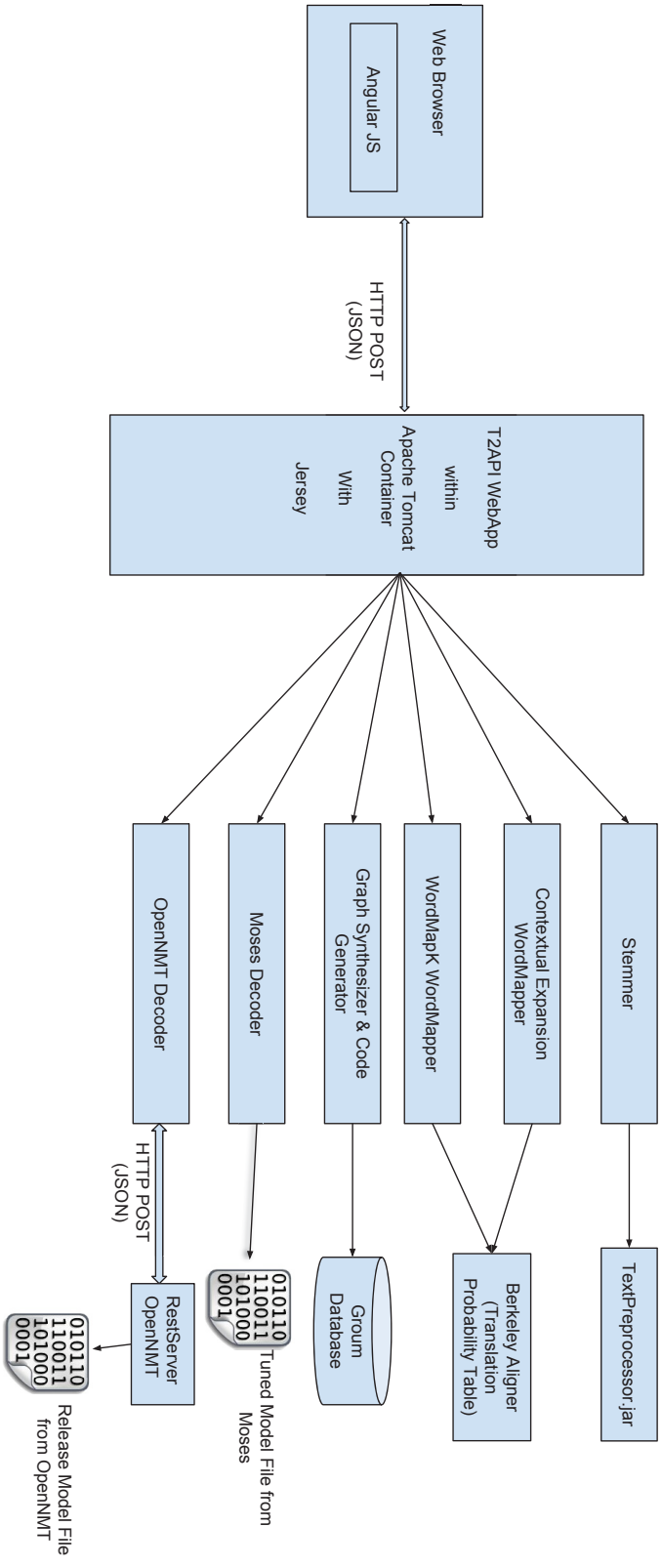


Table 15: English description of software engineering tasks and their stemmed outputs

Raw English Text	Stemmed Output
get location when GPS is not present	get locat gps
decode a bitmap image	decod bitmap imag
read the file line by line and append in a string	read file line line append string
play media file one after another continuously	play media file continu
insert a record into sqllite database	insert record sqllite databas

the file path is provided as argument to the Stemmer executable JAR file. The output file from the stemmer is read and then returned as a JSON response to the client. Table 15 shows the stemmer output for various raw English queries text.

6.3.2 WordMapK Mapper

The WordMapK algorithm is a simple maximum likelihood mapping between each English input word and the most probable K code elements. The translation probability table is derived using the Berkeley Aligner. WordMapK algorithm use the translation probability table and builds the mapping model to translate K code element for each English input text. The translation probability file is loaded during the web application initialisation routine. The WordMapK mapping produces a set of API elements for the query in English text.

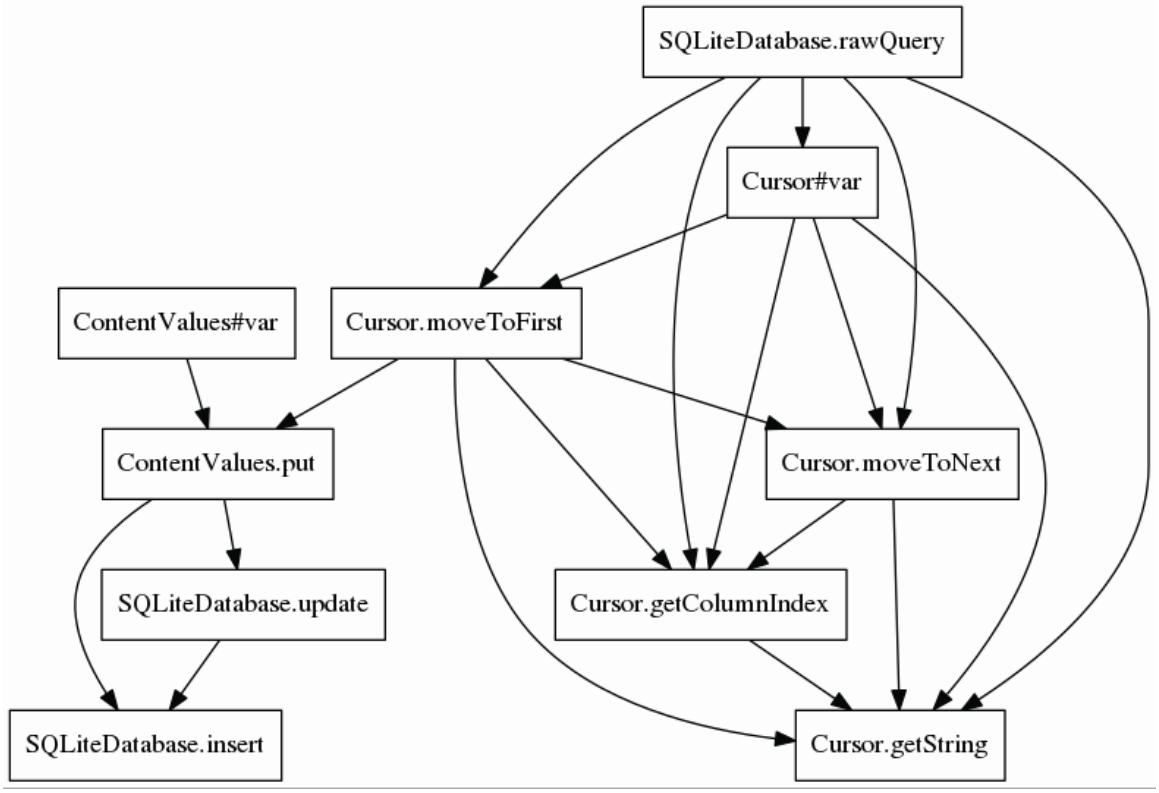
6.3.3 Contextual Expansion Mapper

ContextualExpansion algorithm use the translation probability table from the Berkeley Aligner to build the expansion model. It also computes the co-occurring frequency of English text and source code elements by analysing the training parallel corpora. This process is executed offline and resulting expansion model along with co-occurring frequencies written in a binary file. These files are loaded during the web application initialisation routine. The ContextualExpansion algorithm takes as input the stemmed words and does expansion of API elements. It outputs unordered API elements which are then used by the GraphOrdering component.

6.3.4 Graph Synthesizer

To order the code elements from the word map model, we use *Graph Based API Synthesis (GraSyn)* [61] which takes as input a set of code elements and produces a code graph. A beam search is used to generate the code graph that has the highest probability of occurrence in the training database with

Figure 10: Top ranked Groum for the query "insert a record into sqllite database"



the given input set of code elements from the mapping models. GraSyn will eliminate nodes which make the overall graph less probable. We develope a REST interface to accept the mapped API elements and return a list of API elements along with ranked list of API usage graphs and their corresponding images. Figure 10 gives the top ranked API graph image for the query "insert a record into sqllite database".

6.3.5 Code Generator

We develop the code generator component to translate the Groum (API usage graph) into a usable code template. The GraphOrdering outputs a ranked list of API usage graphs given the list of API elements. The Groum is an abstract representation of the source code with temporal and data dependencies. Providing a graph output would be hard to interpret even for expert software programmers and hence it would be beneficial to transform a given graph into a usable code template. The code generator iterates each connected component of the graph and then identifies the control and data dependencies. For data dependencies it builds the dependency stack in the temporal order API elements from the graph. Then it pops the API elements from the graph and generates the code.

It also does variable creation and assignment for the invocation of constructors or static methods which returns a specific type by inferring the type information of the API elements.

Listing 6.1: CodeGenerator Example

```
Cursor varCursor = SQLiteDatabase.rawQuery();
varCursor.moveToFirst();
ContentValues varContentValues;
varContentValues.put();
varCursor.moveToNext();
SQLiteDatabase.update();
varCursor.getColumnIndex();
SQLiteDatabase.insert();
varCursor.getString();
```

The code listing in 6.1 gives the source code produced by the code generator module for the Figure 10. In this example the groum had only one connected component. The code generator identifies the dependencies and order from the groum and then output the sequence of code elements. In this example the variable varCursor is used declared and assgined to the output of the SQLiteDatabase.rawQuery() method. The T2API objective is to present a set of usable and compilable API elements.

6.3.6 Moses Decoder

The phrase-based machine translation is an extension of word based models with the translation probabilities constructed using phrases (chunks of words). The sequence of code elements are translated from the sequence of English text by mapping the alignment of chunks of text and code elements. The phrase-based machine translation system has implicit reordering capability to generate sequence of words in the target language. We use Moses which is a popular toolkit for phrase based machine translation.

During decoding with Moses decoder, the server will spawn the moses decoder as an external process during startup and provide the location of MERT model file. The moses decoder loads the model file into main memory and waits for input. On an incoming request to decode the English text, the REST interface provides the external moses decoder process with the stemmed query and waits for the decoder process to provide the translations. The moses decoder accepts input and provides output through standard input and output respectively. We use the input and output stream classes from the *Java Development Kit (JDK)* to interact the external process and obtain the translations.

6.3.7 OpenNMT Decoder

OpenNMT uses artificial neural networks and deep learning to perform the sequence to sequence translation. Neural machine translation is the current state-of-the-art for machine translation and is used by Google, Microsoft, and Yandex [12]. The target language sequence prediction is based on complete source sentence and the target sentence that has been produced already. Similar to the phrase based machine translation decoding, we initially tried to spawn the OpenNMT decoder with the trained model and intend to translate the query. However we faced difficulties communicating with OpenNMT decode using JDK input and output stream classes and decided to take the route of using the restserver package that is available as part of OpenNMT toolkit.

The default REST server did not allow *Cross Origin Resource Sharing (CORS)* request for HTTP OPTIONS method. This prevented us from invoking the REST URL of the OpenNMT server directly from the web client. Hence we created a proxy REST end point in T2API web application and rerouted the request to the rest server in OpenNMT. We use the Apache Http Client library to interact with the OpenNMT rest server and transformed the request, response from and to the T2API web client.

6.4 Client Components

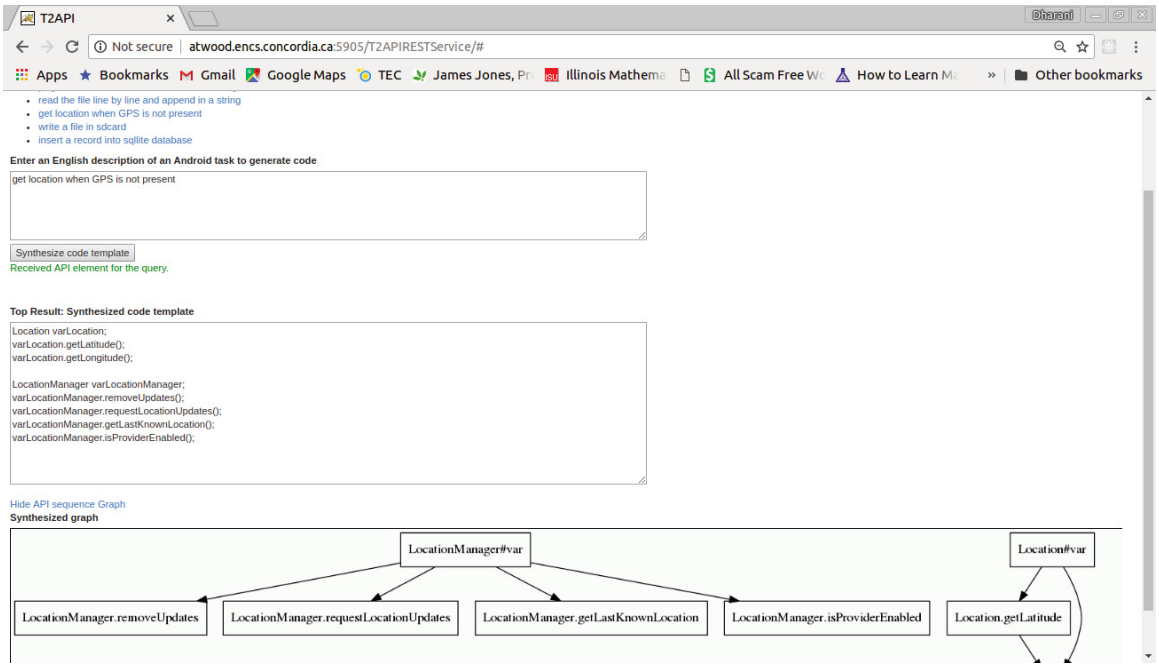
On the client side AngularJS framework is used to communicate with the server as well as to handle user interactions. AngularJS[2] is a Javascript framework which supports model view controller pattern. The view in T2API web portal comprises of the text fields required to capture user query and present them with API code template. The model manages the data shown in these fields and the controller handles the communication with the Apache tomcat server by invoking the REST API end points.

The web application consists of three different views (i.e web pages). Figure 11 shows the interface which accepts an English text in the input field and performs the entire translation using ContextualExpansion algorithm with GraphOrdering. Figure 12 show the complimentary debug interface to educate the user with stage wise execution of translation from English text to API elements with ContextualExpansion algorithm and GraphOrdering. Figure 13 shows the interface to present translation outputs of different machine translation systems for the same English query.

6.4.1 Deployment

The final product is deployed as a *Web application resource or Web application archieve (WAR)* file in the Apache Tomcat container. A config file is available to scaffold the environment and load

Figure 11: Main interface stage wise execution in T2API web portal



the dependencies. More details on the deployment and server configurations are available in the Appendix A.

Figure 12: Debug interface showing stage wise execution in T2API web portal

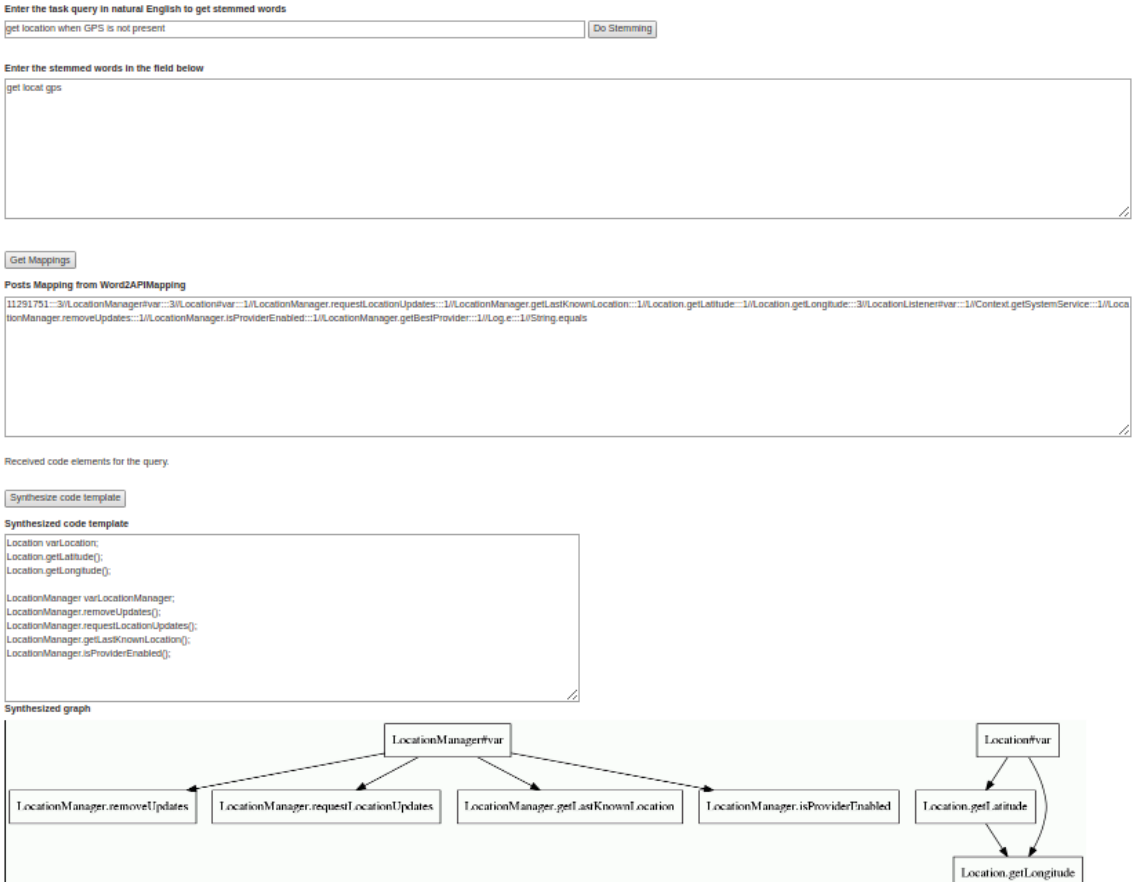
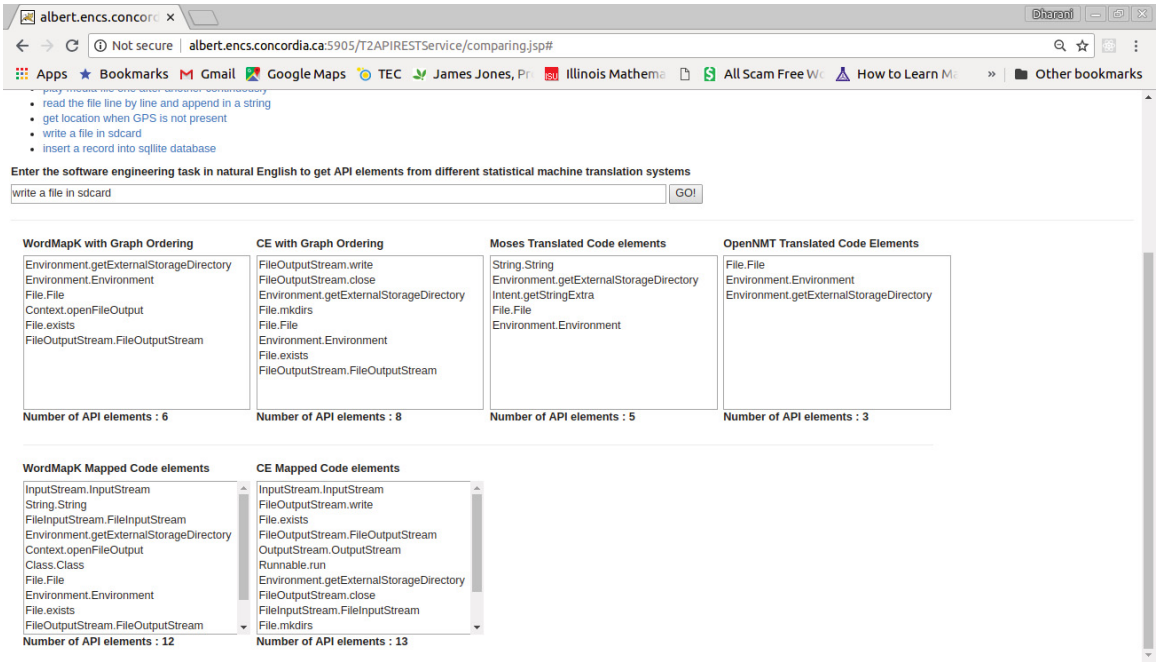


Figure 13: Interface to compare the outputs of different machine translation systemsT2API web portal



Chapter 7

Concluding Remarks

The main contribution of this thesis is an empirical evaluation of four machine translation approaches in the context of software engineering. We evaluated each of the following approaches:

1. WordMapK uses a simple maximum likelihood model to map each English word in the input to K code elements in the output. The elements are then ordered into a graph. The median precision and recall for $K = 5$ is 20 and 50.

2. ContextualExpansion uses the WordMapK model but incorporates the context of commonly co-occurring code elements. The elements are ordered into a graph. The median precision and recall for $K = 5$ is 25 and 40. ContextualExpansion accounts for context which improves the precision at the expense of recall when compared with WordMapK.

3. Phrase-based translation translates sequences of words. As a result, English phrases are mapped to code phrases. The median precision and recall is 25 and 40. Phrase-based translation and ContextualExpansion produces similar translated outputs.

4. Neural networks translate sequences of English to code sequences. The median precision and recall are 100 and 20. Unfortunately, the output is usually limited to a small number of class constructors and typically does not provide interesting API usages.

We conclude that using existing machine translation systems which are designed to translate natural languages shows low precision in translating English text to API sequences without adapting it for the software engineering context. Our study shows that precision and recall values are low in the existing machine translation systems compared to other software engineering specific learning approaches. Our scripts, corpus, and reference translations are available in our replication package [66]. Our CompareSMT web console and REST API is also available.[1] We hope that this work will lead

[1] CompareSMT: `https://users.encs.concordia.ca/~pcr/CompareSMT.html`

to inspire future work that adapts neural networks and other modern machine translation approaches to a software engineering context.

Appendices

Appendix A

Configuration Information

A.1 Berkeley Aligner with fixes

The Berkeley Aligner source code was initially hosted in Google Code Repository (https://code.google.com). After the shutdown of Google code services in March 2015, several independent programmers forked Berkeley Aligner project from Google code and forked them to their personal and public repositories. At this moment there are about 20 different forks of Berkeley Aligner in the famous source code repository GitHub (https://github.com). We have modified the Berkeley Aligner for the following reasons/issues

- The alignment information between English text and source code is written using index of English words instead of the actual terms.
- The default implementation did not support multiple threads.
- A bug, that prevented the translation probability table being written completely on disk when processed for large parallel corpora.

Hence to train the parallel corpora for word alignment with Berkeley Aligner it is recommended to use the modified Berkeley Aligner that is available in our replication package [66]

Follow the instructions on the README in the root directory of the Berkeley Aligner project to compile and generate the "berkeleyaligner.jar" file in the distribution directory.

A.2 OpenNMT Training - Server Configuration

OpenNMT is a neural machine translation toolkit. For training the English text to source code parallel corpora we require *Graphics Processing Unit (GPU)*. We trained our corpora using Google

cloud infrastructure.

We setup a virtual machine in the Google cloud infrastructure with 4 *Nvidia K80* GPU. The encoder layer of the RNN has 4 layers and the decoder has 8 layers. The size of the neural net is 512 nodes. Each graphics processing unit had about 12 GB of memory and the server had 8 virtual CPU cores with 30 GB of memory. The training was configured for 30 epochs with each epoch spanning $10,000$ iterations totalling approximately 300K iterations. This configuration would approximately takes about 4 days to complete the training with the parallel corpora.

To install OpenNMT toolkit, follow the installation instructions provided in the URL http://opennmt.net/OpenNMT/installation/.

In our setup we did not use the Docker container. You need to install the REST server of OpenNMT toolkit to execute the decoder. To install the REST server of OpenNMT toolkit follow the instructions provided at http://opennmt.net/OpenNMT/tools/servers/

If the training happened on GPUs then you might have to convert the trained model to a release model file in the OpenNMT if you intend to run the OpenNMT decoder in a non GPU environment. In order to convert the trained model to a release model follow the instructions provided at http://opennmt.net/OpenNMT/options/release_model/

A.3 Moses - Server Configuration

Moses is the phrase based machine translation toolkit. Moses requires adequate main memory and more cores to expedite the training process. In our study we ran Moses on server with 24GB of main memory and 8 logical cores. Moses should be configured to use *Multithreaded Giza (mgiza)* to leverage the available cores. We followed the instructions provided in the Moses official website (http://www.statmt.org/moses/?n=Development.GetStarted) to setup and configure the toolkit.

We faced issues with tokenization script provided as part of Moses toolkit, because our source code corpus had API elements separated by "." character. Hence for Moses alone we trained by converting the dot character to underscore character. For example, the API element `String.substring` was converted to `String_substring` to mitigate this problem.

A.4 T2API Web Portal

T2API Web portal is a Apache Tomcat web application. To setup the T2API Web portal install the software components shown in the Table 16

Table 16: Software Requirements for T2API

Library	Version or Location
Eclipse IDE	Neon version or later with JEE support
Apache Tomcat	8.5.0
Jersey	1.20-SNAPSHOT
Apache Http Client	4.5.6
Java Development Kit	1.8.0_101
Moses	https://github.com/moses-smt/mosesdecoder
OpenNMT	https://github.com/OpenNMT/OpenNMT
Torch	http://torch.ch/docs/getting-started.html

A.5 T2API Deployment

The T2API Web portal is a Apache Maven project and includes a *Project Object Model (POM)*. Import the project using the Eclipse import option. The project will get build and all the dependencies will be resolved. You will have to setup a runTime environment to point to Apache Tomcat installation directory. To generate the *Web ARchieve or Web Application Resource (WAR)* file, choose "File –> Export" option, then select WAR option under "Web" menu. Enter a destination directory and generate the WAR file. Deploy the WAR file to the "webapps" directory under the Apache Tomcat installation directory. There is a "config.properties" file available in the project which contains key-value pairs to configure the path.

Table 17 shows the list of configuration keys and their descriptions.

A.5.1 Moses And OpenNMT Decoders

To execute the decoder components of Moses and OpenNMT toolkits, update the "config.properties" file with relevant information by referring to Table 17.

The Moses decoder will be invoked by the web application when there is a request to translate code elements using Moses. The OpenNMT REST server has to be started before starting the Apache Tomcat server that hosts the T2API web application. The command to start the OpenNMT REST server is provided in the Listing A.1. Replace the path argument with the path of the release model file.

Listing A.1: OpenNMT Decoder using REST Server

```
$ th tools/rest_translation_server.lua -model -host 0.0.0.0 -model <path release model>
```

Table 17: Deployment Configuration for T2API

Configuration Key	**Description of Key and Value to set**
BASE_DIR	Sandbox directory to use for the web application
JAVA_EXEC_PATH	Path of the Java executable to invoke the Stemmer
STEMMER_JAR_LOCATION	Path of the Stemmer JAR file. We have included the Jar file in the [66]
GRALAN_DATABASE_DIR	Location of the graph database
WORD_MAPPING_CONFIG_DIR, WORD_MAPPING_BERK_DIR	Directory containing configuration files for ContextualExpansion algorithm. Available in [66]
MOSES_BINARY	Full path to the moses executable file. Typically found under the bin directory of the Moses installation.
MERT_WORKING_MOSESINI	Full path to the trained and tuned moses model file.
STAGE_PARAMS_FILE	Translation probability file in the Berkeley Aligner. Typically in the output directory with name "stage2.2.params.txt"
K	K value for WordMapK Algorithm
OPENNMT_REST_SERVER_URL	URL of the OpenNMT REST server

Table 18: Hardware Requirements for T2API

Component	Memory
Apache Tomcat with Graph database, ContextualExpansion Mappings, and WordMapK Mappings	16 GB
Moses	6 GB

Update the JAVA_HOME variable in the file "setenv.sh" in the bin directory of the Apache Tomcat to point to the installation directory of Java Development Kit. Update the CATALINA_OPTS variable to values "-Xms10g -Xmx16g" to set the minimum heap and maximum heap to 10 and 16 GB respectively.

The commands to start and stop the Apache Tomcat is provided in the Listings A.2 and A.3. Change to the Apache Tomcat installation bin and execute the following commands.

Listing A.2: Apache Tomcat Start Command

```
$ ./startup.sh
```

Listing A.3: Apache Tomcat Stop Command

```
$ ./shutdown.sh
```

A.5.2 T2API Web Portal - Server Requirements

The server configuration requirements are listed in Table 18. A server with adequate main memory is required to execute T2API web portal with less latency.

We need a server with a minimum main memory of 24 GB to configure and run the T2API web portal. Higher memory will help to reduce latency and support more concurrent requests.

Appendix B

API Translations from WordMapK, ContextualExpansion Moses and OpenNMT

We give the translation outputs of WordMapK, ContextualExpansion, Phrase based machine translation and Neural machine translation systems for the set of input queries from the validation data set. For WordMapK and ContextualExpansion algorithms we give the API elements with GraphOrdering. The validation data set consists of 240 queries of English text and reference translations. We include the output translations from different machine translation systems in our replication package [66]. In this Section we show 10 translations for each scheme.

Table 19: Translations from WordMapK with GraphOrdering for $K = 5$

English Task	**API Elements**
insert a record into sqllite database	SQLiteDatabase.insert Cursor.Cursor Cursor.getColumnIndex String.String Cursor.moveToNext ContentValues.put ContentValues.ContentValues Cursor.getString
write a file in sdcard	Environment.getExternalStorageDirectory Environment.Environment File.File File.exists Context.openFileOutput FileOutputStream.FileOutputStream
get date and time-of-day	Calendar.getInstance Calendar.Calendar LayoutInflater.inflate View.View Calendar.get Calendar.set
read a JSON response in an array	BufferedReader.BufferedReader BufferedReader.readLine String.String JSONObject.getString JSONArray.getJSONObject JSONArray.length
display html document in a view	WebView.loadUrl View.View Html.fromHtml TextView.TextView Html.Html WebView.loadData View.findViewById WebView.WebView TextView.setText
how to zero the time of a calendar	Calendar.Calendar View.View View.getHeight ContentResolver.insert String.String String.format ContentValues.put Calendar.getTimeInMillis
how to hide virtual keyboard on touch of a spinner	View.View View.setVisibility MotionEvent.MotionEvent ArrayAdapter.createFromResource ArrayAdapter.setDropDownViewResource MotionEvent.getAction ArrayAdapter.ArrayAdapter Spinner.Spinner InputMethodManager.hideSoftInputFromWindow InputMethodManager.InputMethodManager Spinner.setAdapter MotionEvent.getX MotionEvent.getY
Resize a bitmap	Bitmap.Bitmap Bitmap.createBitmap Bitmap.getWidth Bitmap.createScaledBitmap Bitmap.getHeight Canvas.drawBitmap
check internet connection	ConnectivityManager.ConnectivityManager ConnectivityManager.getActiveNetworkInfo NetworkInfo.NetworkInfo Context.getSystemService NetworkInfo.isConnected
hide soft keyboard	View.View View.setVisibility InputMethodManager.hideSoftInputFromWindow InputMethodManager.InputMethodManager

Table 20: Translations from ContextualExpansion with GraphOrdering for $K = 5$

English Task	API Elements
insert a record into sqllite database	SQLiteDatabase.insert SQLiteDatabase.update Cursor.Cursor Cursor.getColumnIndex SQLiteDatabase.rawQuery Cursor.moveToNext ContentValues.ContentValues ContentValues.put Cursor.moveToFirst Cursor.getString
write a file in sdcard	File.getAbsolutePath FileOutputStream.write FileOutputStream.close Environment.getExternalStorageDirectory Environment.Environment File.File File.mkdirs Context.openFileOutput FileOutputStream.FileOutputStream File.exists File.delete File.createNewFile
get date and time-of-day	Calendar.Calendar Calendar.getInstance Calendar.get String.equals Calendar.getTime Calendar.set Calendar.getTimeInMillis Integer.parseInt
read a JSON response in an array	HttpResponse.HttpResponse HttpResponse.getEntity JSONObject.getString JSONArray.length JSONArray.getString JSONObject.getJSONArray
display html document in a view	WebView.loadUrl WebView.getSettings WebView.loadDataWithBaseURL WebView.loadData WebView.WebView
how to zero the time of a calendar	Calendar.getInstance Calendar.Calendar Calendar.get Calendar.set
how to hide virtual keyboard on touch of a spinner	InputMethodManager.hideSoftInputFromWindow InputMethodManager.showSoftInput MotionEvent.MotionEvent InputMethodManager.InputMethodManager MotionEvent.getAction MotionEvent.getX
Resize a bitmap	ImageView.ImageView Bitmap.createBitmap Canvas.Canvas Bitmap.getWidth BitmapFactory.decodeFile BitmapFactory.BitmapFactory BitmapFactory.decodeResource Bitmap.getHeight ImageView.setScaleType ImageView.setImageBitmap ImageView.setImageResource Canvas.drawBitmap
check internet connection	ConnectivityManager.ConnectivityManager ConnectivityManager.getActiveNetworkInfo Context.getSystemService NetworkInfo.NetworkInfo NetworkInfo.isConnected
hide soft keyboard	InputMethodManager.hideSoftInputFromWindow InputMethodManager.showSoftInput InputMethodManager.InputMethodManager

Table 21: Translations from Moses - Phrase based machine translation toolkit

English Task	**API Elements**
insert a record into sqllite database	MediaRecorder.prepare SQLiteDatabase.insert String.String ContentValues.ContentValues ContentValues.put
write a file in sdcard	String.String Environment.getExternalStorageDirectory Intent.getStringExtra File.File Environment.Environment
get date and time-of-day	Calendar.Calendar Calendar.getInstance SimpleDateFormat.SimpleDateFormat String.String Calendar.getTime Activity.Activity Date.Date
read a JSON response in an array	String.String JSONArray.JSONArray JSONObject.JSONObject JSONObject.getString JSONArray.length JSONArray.getJSONObject JSONObject.getJSONArray
display html document in a view	View.View Html.fromHtml Activity.Activity Html.Html Spanned.Spanned
how to zero the time of a calendar	Calendar.Calendar View.View String.substring
how to hide virtual keyboard on touch of a spinner	View.View Context.Context InputMethodManager.hideSoftInputFromWindow Context.getSystemService MotionEvent.MotionEvent InputMethodManager.InputMethodManager MotionEvent.getAction MotionEvent.getX MotionEvent.getY Spinner.Spinner
Resize a bitmap	Bitmap.Bitmap Bitmap.getHeight
check internet connection	Context.Context ConnectivityManager.ConnectivityManager ConnectivityManager.getActiveNetworkInfo Context.getSystemService NetworkInfo.NetworkInfo NetworkInfo.isConnectedOrConnecting
hide soft keyboard	View.View Context.Context Context.getSystemService InputMethodManager.hideSoftInputFromWindow InputMethodManager.showSoftInput EditText.EditText InputMethodManager.InputMethodManager

Table 22: Translations from OpenNMT - Neural machine translation toolkit

English Task	**API Elements**
insert a record into sqllite database	ContentValues.ContentValues ContentValues.put
write a file in sdcard	File.File Environment.Environment Environment.getExternalStorageDirectory
get date and time-of-day	Calendar.Calendar Calendar.getInstance Calendar.get
read a JSON response in an array	JSONArray.JSONArray JSONObject.JSONObject
display html document in a view	WebView.WebView
how to zero the time of a calendar	Calendar.Calendar Calendar.getInstance Calendar.get
how to hide virtual keyboard on touch of a spinner	InputMethodManager.InputMethodManager Context.Context Context.getSystemService InputMethodManager.hideSoftInputFromWindow
Resize a bitmap	Bitmap.Bitmap Bitmap.createScaledBitmap
check internet connection	ConnectivityManager.ConnectivityManager Context.Context NetworkInfo.NetworkInfo Context.getSystemService ConnectivityManager.getActiveNetworkInfo NetworkInfo.isConnected
hide soft keyboard	InputMethodManager.InputMethodManager Context.Context Context.getSystemService InputMethodManager.hideSoftInputFromWindow

Bibliography

[1] About stack overflow, 2018. Retrived from `https://stackoverflow.com/company`.

[2] Angular js, 2018. Retrieved from `https://angularjs.org/`.

[3] Apache tomcat, 2018. Retrieved from `http://tomcat.apache.org/`.

[4] Jersey restful web services in java, 2018. Retrieved from `https://jersey.github.io/`.

[5] Snowball stemmer, 2018. Retrieved from `http://snowballstem.org/`.

[6] M. Acharya, T. Xie, J. Pei, and J. Xu. Mining api patterns as partial orders from source code: from usage scenarios to specifications. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 25–34. ACM, 2007.

[7] A. Alnusair, T. Zhao, and E. Bodden. Effective api navigation and reuse. In *Information Reuse and Integration (IRI), 2010 IEEE International Conference on*, pages 7–12. IEEE, 2010.

[8] D. L. Atkins. Version sensitive editing: Change history as a programming tool. In *International Workshop on Software Configuration Management*, pages 146–157. Springer, 1998.

[9] S. Azad, P. C. Rigby, and L. Guerrouj. Generating api call rules from version history and stack overflow posts. *ACM Trans. Softw. Eng. Methodol.*, 25(4):29:1–29:22, Jan. 2017.

[10] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[11] S. Banerjee and A. Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, volume 29, pages 65–72, 2005.

[12] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, M. Huck, A. J. Yepes, P. Koehn, V. Logacheva, C. Monz, et al. Findings of the 2016 conference on machine translation. In *ACL 2016 FIRST CONFERENCE ON MACHINE TRANSLATION (WMT16)*, pages 131–198. The Association for Computational Linguistics, 2016.

[13] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[14] P. F. Brown, J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

[15] P. F. Brown, J. C. Lai, and R. L. Mercer. Aligning sentences in parallel corpora. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 169–176. Association for Computational Linguistics, 1991.

[16] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[17] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm. Eye movements in code reading: Relaxing the linear order. In *Program Comprehension (ICPC), 2015 IEEE 23rd International Conference on*, pages 255–265. IEEE, 2015.

[18] B. A. Campbell and C. Treude. Nlp2code: Code snippet content assist via natural language tasks. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, pages 628–632. IEEE, 2017.

[19] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[21] L. D. Consortium et al. Hansard corpus of parallel english and french, 1997.

[22] M. E. Crosby, J. Scholtz, and S. Wiedenbeck. The roles beacons play in comprehension for novice and expert programmers. In *14th Workshop of the Psychology of Programming Interest Group*, pages 58–73, 2002.

[23] M. E. Crosby and J. Stelovsky. How do we read algorithms? a case study. *Computer*, 23(1):25–35, 1990.

[24] D. Čubranić and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th international Conference on Software Engineering*, pages 408–418. IEEE Computer Society, 2003.

[25] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[26] J. DeNero and D. Klein. Tailoring word alignments to syntactic machine translation. In *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, volume 45, page 17, 2007.

[27] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[28] Dzmitry Bahdanau, Kyunghyun Cho. Groundhog - recurrent neural network, 2014. `https://github.com/lisa-groundhog/GroundHog`.

[29] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[30] X. Gu, H. Zhang, D. Zhang, and S. Kim. Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642. ACM, 2016.

[31] K. Heafield. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197. Association for Computational Linguistics, 2011.

[32] K. M. Hermann and P. Blunsom. Multilingual distributed representations without word alignment. *arXiv preprint arXiv:1312.6173*, 2013.

[33] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 837–847. IEEE, 2012.

[34] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[35] R. Holmes, R. J. Walker, and G. C. Murphy. Strathcona example recommendation tool. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 237–240. ACM, 2005.

[36] M. Hopkins and J. May. Tuning as ranking. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362. Association for Computational Linguistics, 2011.

[37] J. Jones. Abstract syntax tree implementation idioms. In *Proceedings of the 10th conference on pattern languages of programs (plop2003)*, pages 1–10, 2003.

[38] S. Karaivanov, V. Raychev, and M. Vechev. Phrase-based statistical translation of programming languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, pages 173–184. ACM, 2014.

[39] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv e-prints*.

[40] P. Koehn. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Conference of the Association for Machine Translation in the Americas*, pages 115–124. Springer, 2004.

[41] P. Koehn et al. Europarl: A multilingual corpus for evaluation of machine translation, 2002.

[42] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[43] P. Koehn, F. J. Och, and D. Marcu. 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*, pages 48–54. Association for Computational Linguistics, 2003.

[44] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, R. S. Morla, P. C. Masiero, P. Baldi, and C. V. Lopes. Codegenie: using test-cases to search and reuse source code. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 525–526. ACM, 2007.

[45] Z. Li and Y. Zhou. Pr-miner: automatically extracting implicit programming rules and detecting violations in large software code. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 306–315. ACM, 2005.

[46] P. Liang, B. Taskar, and D. Klein. Alignment by agreement. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of*

the Association of Computational Linguistics, pages 104–111. Association for Computational Linguistics, 2006.

[47] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk. Mining energy-greedy api usage patterns in android apps: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 2–11. ACM, 2014.

[48] G. Little and R. C. Miller. Keyword programming in java. *Automated Software Engineering*, 16(1):37, 2009.

[49] C. Liu, E. Ye, and D. J. Richardson. Software library usage pattern extraction using a software model checker. *International Journal of Computers and Applications*, 31(4):247–259, 2009.

[50] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[51] W. Macherey, F. J. Och, I. Thayer, and J. Uszkoreit. Lattice-based minimum error rate training for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 725–734. Association for Computational Linguistics, 2008.

[52] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[53] A. Michail. Data mining library reuse patterns in user-selected applications. In *Automated Software Engineering, 1999. 14th IEEE International Conference on.*, pages 24–33. IEEE, 1999.

[54] A. Michail. Data mining library reuse patterns using generalized association rules. In *Proceedings of the 22nd international conference on Software engineering*, pages 167–176. ACM, 2000.

[55] Y. M. Mileva, V. Dallmeier, and A. Zeller. Mining api popularity. In *Testing–Practice and Research Techniques*, pages 173–180. Springer, 2010.

[56] Moses Support. Moses hardware requirements, 2016. Retrieved from `http://www.mail-archive.com/moses-support@mit.edu/msg14821.html`.

[57] G. Neubig. lamtram: A toolkit for language and translation modeling using neural networks, 2015.

[58] G. Neubig and T. Watanabe. Optimization for statistical machine translation: A survey. *Computational Linguistics*, 42(1):1–54, 2016.

[59] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen. Grapacc: a graph-based pattern-oriented, context-sensitive code completion tool. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1407–1410. IEEE Press, 2012.

[60] T. Nguyen, P. C. Rigby, A. T. Nguyen, M. Dharani, Palani ano Karanfil, and T. N. Nguyen. Statistical translation of english texts to api code templates. In *Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution*, ICSME '18, Washington, DC, USA, 2018. IEEE Computer Society.

[61] T. Nguyen, P. C. Rigby, A. T. Nguyen, M. Karanfil, and T. N. Nguyen. T2api: Synthesizing api code usage templates from english texts with statistical translation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 1013–1017. ACM, 2016.

[62] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen. A statistical semantic language model for source code. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 532–542. ACM, 2013.

[63] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen. Graph-based mining of multiple object usage patterns. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 383–392. ACM, 2009.

[64] F. J. Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics, 2003.

[65] F. J. Och and H. Ney. Giza++: Training of statistical translation models, 2000.

[66] D. K. Palani. Replication package, 2018. Retrived from `https://github.com/CESEL/EnglishToCodeSMT`.

[67] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[68] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 102–111. ACM, 2014.

[69] M. Raghothaman, Y. Wei, and Y. Hamadi. Swim: Synthesizing what i mean-code search and idiomatic snippet synthesis. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 357–367. IEEE, 2016.

[70] M. Rahman. *Analyzing the Predictability of Source Code and its Application in Creating Parallel Corpora for English-to-Code Statistical Machine Translation*. PhD thesis, Concordia University, 2018.

[71] S. P. Reiss. Semantics-based code search. In *Proceedings of the 31st International Conference on Software Engineering*, pages 243–253. IEEE Computer Society, 2009.

[72] P. C. Rigby and M. P. Robillard. Discovering essential code elements in informal documentation. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 832–841. IEEE Press, 2013.

[73] M. P. Robillard. What makes apis hard to learn? answers from developers. *IEEE software*, 26(6), 2009.

[74] M. P. Robillard and R. Deline. A field study of api learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

[75] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D'Mello. Improving automated source code summarization via an eye-tracking study of programmers. In *Proceedings of the 36th International Conference on Software Engineering*, pages 390–401. ACM, 2014.

[76] N. Sahavechaphan and K. Claypool. Xsnippet: Mining for sample code. *ACM Sigplan Notices*, 41(10):413–430, 2006.

[77] C. Scaffidi. Why are apis difficult to learn and use? *Crossroads*, 12(4):4–4, 2006.

[78] A. G. Schuth and C. Monz. Tuning methods in statistical machine translation. 2010.

[79] H. Schwenk. Continuous space language models. *Computer Speech & Language*, 21(3):492–518, 2007.

[80] C. E. Shannon. A mathematical theory of communication (parts i and ii). *Bell System Tech. J.*, 27:379–423, 1948.

[81] J. Stylos and B. A. Myers. Mica: A web-search tool for finding api components and examples. In *null*, pages 195–202. IEEE, 2006.

[82] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[83] E. Thiébaut. Mira: an effective imaging algorithm for optical interferometry. In *Optical and Infrared Interferometry*, volume 7013, page 70131I. International Society for Optics and Photonics, 2008.

[84] S. Thummalapenta and T. Xie. Parseweb: a programmer assistant for reusing open source code on the web. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 204–213. ACM, 2007.

[85] G. Uddin, B. Dagenais, and M. P. Robillard. Analyzing temporal api usage patterns. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 456–459. IEEE, 2011.

[86] J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang. Mining succinct and high-coverage api usage patterns from source code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 319–328. IEEE Press, 2013.

[87] Wikipedia. Microsoft translator — Wikipedia, the free encyclopedia, 2018. Retrived from `http://en.wikipedia.org/wiki/Microsoft_Translator`.

[88] Wikipedia. Stack overflow — Wikipedia, the free encyclopedia, 2018. Retrived from `https://en.wikipedia.org/wiki/Stack_Overflow`.

[89] D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[90] T. Xie and J. Pei. Mapo: Mining api usages from open source repositories. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 54–57. ACM, 2006.

[91] Y. Ye. Programming with an intelligent agent. *IEEE Intelligent Systems*, 18(3):43–47, 2003.

[92] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE transactions on Software Engineering*, 30(9):574–586, 2004.