Prefix Resolution

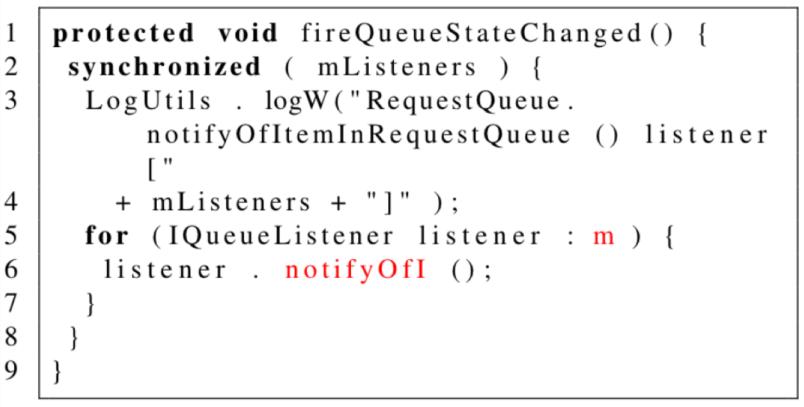
Hung PHAN

Overview

Machine Translation is a potential approach for solving problems in Natural Language and Software Engineering. In Machine Translation (MT) techniques, Neural Machine Translation (NMT) is considered as the newest trend which outperforms the state of the art approach, Statistical Machine Translation (SMT) in Natural Language Translation thanks to its ability to take advantages of Deep Learning. However, prior works show that in problems that the parallel corpus has two special characteristics, the SMT achieved higher accuracy compared to SMT in BLEU score and in word-level accuracy comparison.These two characteristics are the consistent between length of source-target pairs and consistent between order of source tokens and target tokens. In this work, we explore a research problem which has these characteristics, that is how to map from space of prefixes/ abbreviation of words with short sequences of letters appeared in each word to the space of words. We build Machine Translation engines based on SMT and NMT to learn the mapping for words/ code tokens generation from 3 types of corpus: English general text, software documentation and programming language. In these corpus, we analyze the ability of mapping from prefixes in different type of text. We focus our work on the programming language corpus and propose the application of MT in code tokens suggestion from prefix or abbreviation. By the evaluation, we show that the SMT outperforms NMT in this research problem, which provides potential direction to improve the current NMT engines to be optimize in specific classes of parallel corpus. By achieving the accuracy from 65% to 90% for code tokens generation of 1000 Github code corpus, we show the potential of using MT for code completion at token level.

Input & Output

Input

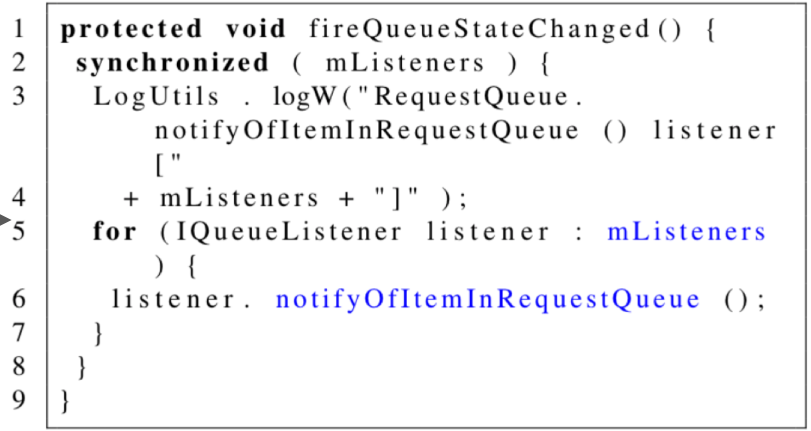


```
protected void fireQueueStateChanged() {
 synchronized ( mListeners ) {
  LogUtils . logW("RequestQueue.
      notifyOfItemInRequestQueue () listener
      ["
    + mListeners + "]" );
  for (IQueueListener listener : m ) {
   listener . notifyOfI ();
  }
 }
}
```

PrefixMapping

Code Editor

Output

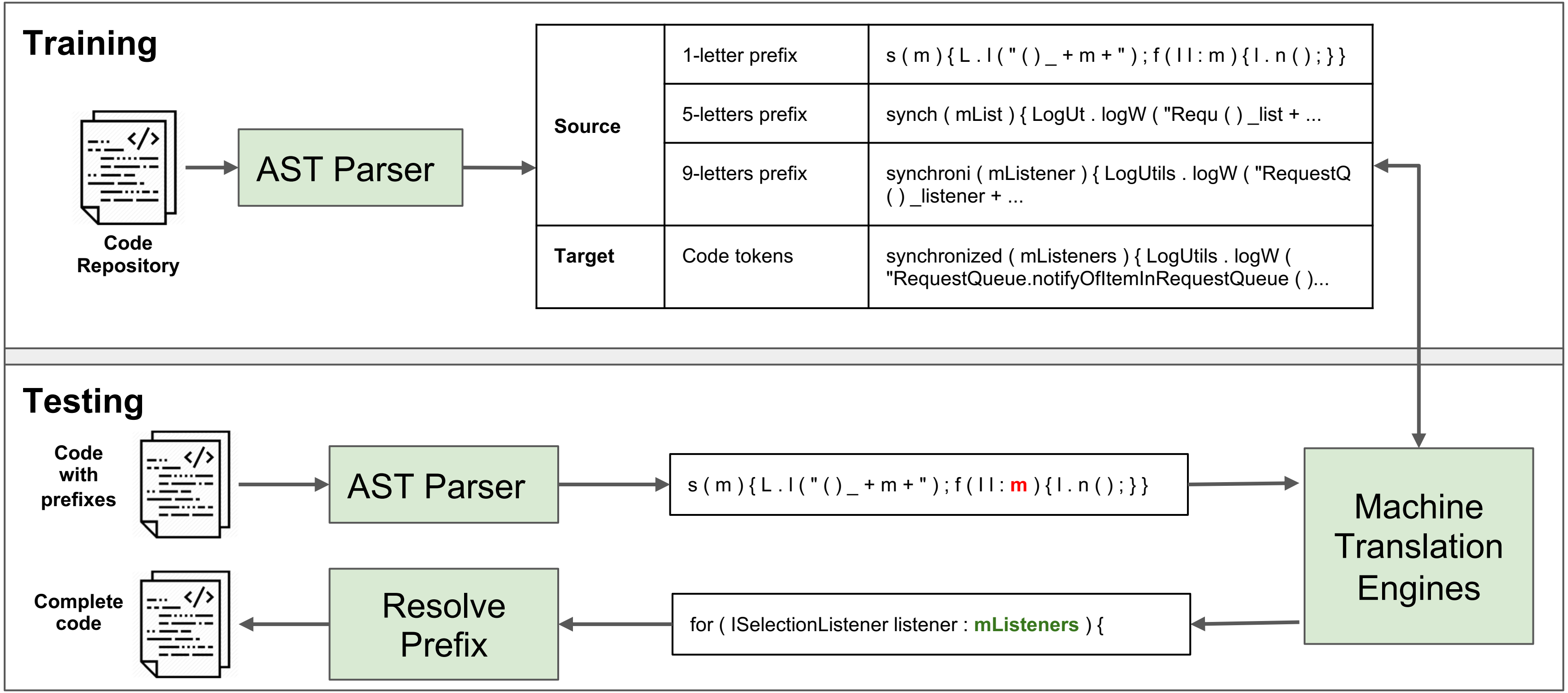


```
protected void fireQueueStateChanged() {
 synchronized ( mListeners ) {
  LogUtils . logW("RequestQueue.
      notifyOfItemInRequestQueue () listener
      ["
    + mListeners + "]" );
  for (IQueueListener listener : mListeners
      ) {
   listener. notifyOfItemInRequestQueue ();
  }
 }
}
```

Advantages:

1. Prefix = first letters of code tokens.
2. Developers only write first letter of tokens to get output.
3. Suggest all types of code tokens.

How PrefixMapping works



Training

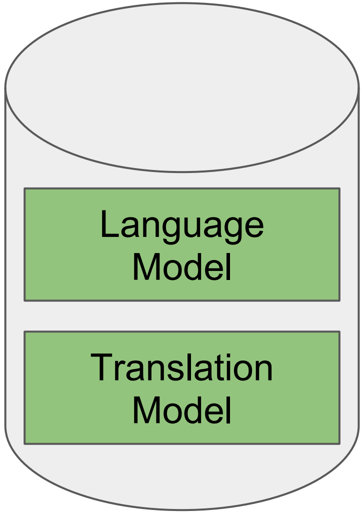
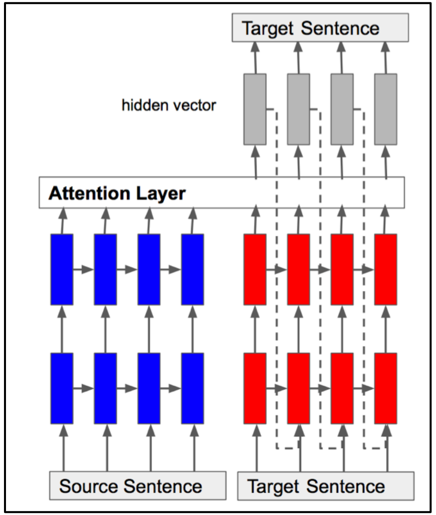
Source	1-letter prefix	s (m) { L . l (" ( ) _ + m + ") ; f (l l : m) { l . n ( ) ; } }
	5-letters prefix	synch (mList) { LogUt . logW ("Requ ( ) _list + ...
	9-letters prefix	synchroni (mListener) { LogUtils . logW ("RequestQ ( ) _listener + ...
Target	Code tokens	synchronized (mListeners) { LogUtils . logW ("RequestQueue.notifyOfItemInRequestQueue ( )...

Testing

s (m) { L . l (" ( ) _ + m + ") ; f (l l : **m**) { l . n ( ) ; } }

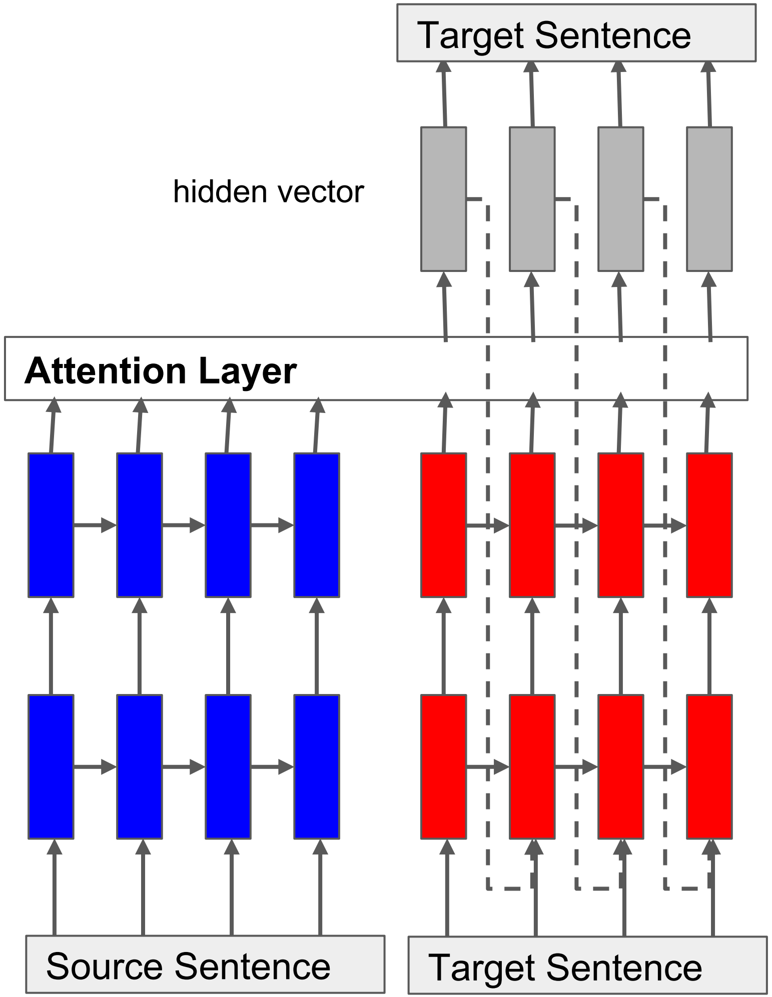
for (ISelectionListener listener : **mListeners**) {

Machine Translation Engines

Statistical Machine Translation	Neural Machine Translation
	 Target Sentence hidden vector Attention Layer Source Sentence Target Sentence
Precision: 65% (1 letter) to 90% (9-letters)	**Precision: 58% (1 letter) to 82% (9-letters)**

Contributions

1. Provide a tool of code completion for prefixes in the form of any types of code tokens.
2. Generalize a class of research problems which Neural Machine Translation outperforms Statistical Machine Translation.
3. Summarize metrics for evaluating this class of problems using translation.
4. Analysis on prefix mapping depending on different types of Programming Language elements.



Target Sentence

hidden vector

Attention Layer

Source Sentence

Target Sentence