Poster: Comparing Neural Network Solutions in Cryptographic API Completion

Ya Xiao*, Salman Ahmed*, Wenjia Song*, Bimal Viswanath*, Na Meng*, Patrick McDaniel†,
Xinyang Ge‡, and Danfeng (Daphne) Yao*
Department of Computer Science, Virginia Tech, Blacksburg, VA*
Department of Computer Science and Engineering, Penn State University, State College, PA†
Microsoft Research, Redmond, WA‡
{yax99,ahmedms,wenjia7,vbimal,nm8247,danfeng}@vt.edu, mcdaniel@cse.psu.edu, aegiryy@gmail.com

Abstract—With the strong interest in neural network based software engineering approaches and a plethora of proposed solutions, we point out the need for measurement studies in this space. Focusing on a specific application scenario, Java cryptographic API code completion, we outline several potential measurement problems, ranging from embedding design and evaluation, to methodology development of models, and to metrics and benchmarks.

I. INTRODUCTION

The attractive vision of automatic code engineering, e.g., repair [5], [8] and generation [11], [11], has motivated a line of neural networks based machine learning solutions [1], [2]. Given the tremendous success in natural language processing, it is conceivable that deep learning has the potential to revolutionize how code is generalized, transformed, and patched.

In this project, we focus on a specific application scenario, Cryptographic API completion. Cryptographic APIs are reported to be error-prone and result in many security vulnerabilities that seriously threaten software security [10]. We systematically measured the accuracy impacts of the state-of-the-art neural network solutions for cryptographic API completion [12]. The neural network solutions include two key steps, representing programs as numeric vectors, and training a neural network on these vectors. Therefore, our experiments compare different choices for the vectorization, aka code embedding, and the neural networks. We further performed in-depth manual analysis to uncover the unreported challenges from programming language-specific properties.

Comparisons of program analysis guided embeddings. For neural network based approaches, programs need to be first represented as vectors to feed into neural networks. Code embedding refers to the process of automatically learning the low-dimensional vector representations of program elements [2], [7]. Intuitively, it is about how to meaningfully express code in vectors. This transformation is important, as subsequent tasks are performed on the embeddings of code.

Despite recent progress [2], [7], [3], there has not been any systematic investigation of various code embedding designs or comprehensive evaluation in terms of their security and accuracy capabilities. Such side-by-side comparisons would help better design neural network based methodologies and harness their power for code embedding approaches.

We conducted a comprehensive comparison to learn the impacts of program analysis guidance on the quality of code embedding. By applying program analysis preprocessing, the code sequence can be transformed into more structural representations. These structural representations can provide more meaningful context information for code embedding. As shown in Fig. 1 (a) shows the API sequences extracted from byte code while (b) and (c) display API sequences of program slices and API dependence graphs that are obtained by program analysis, respectively. We apply skip-gram embedding model [2] on the byte code, slices, and dependence paths extracted from the API dependence graphs, respectively, to produce three types of API embeddings, byte2vec, slice2vec, and dep2vec. These embeddings, as well as a basic one-hot vector baseline, are used as the inputs when training LSTM based models for cryptographic API completion tasks.

TABLE I: Accuracy of next API Recommendation.

<table>
<thead>
<tr>
<th>LSTM Units</th>
<th>Byte Code</th>
<th>Slices</th>
<th>Dependence Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-hot</td>
<td>byte2vec</td>
<td>1-hot</td>
</tr>
<tr>
<td>64</td>
<td>49.78%</td>
<td>48.31%</td>
<td>53.91%</td>
</tr>
<tr>
<td>128</td>
<td>53.01%</td>
<td>53.52%</td>
<td>53.91%</td>
</tr>
<tr>
<td>256</td>
<td>54.91%</td>
<td>54.59%</td>
<td>70.35%</td>
</tr>
<tr>
<td>512</td>
<td>55.80%</td>
<td>55.96%</td>
<td>71.78%</td>
</tr>
</tbody>
</table>

Table I shows the accuracy of the cryptographic API completion task trained with different embedding settings and LSTM models. We have three comparison groups. First, we compare three types of embeddings, byte2vec, slice2vec, and dep2vec, which are trained on different program analysis preprocessed corpora, byte code, slices, and dependence paths, respectively. Second, we also compare the embedding option with its one-hot baseline on each type of code corpus. Moreover, we compare different sizes of LSTM models in this task. An important observation is that program analysis brings significant benefits, improving the accuracy of cryptographic API completion from 55.96% (with byte2vec) to 92.04% (with dep2vec). We also found that embedding options, slice2vec and dep2vec significantly improve the accuracies by 12% and 6%, compared with their one-hot baselines.

Analysis on programming language-specific challenges. Besides program analysis guided embedding, the neural network design is another important aspect of the API completion solution. Although many neural language models (e.g. LSTM,
Comparisons of specialized neural network designs. We present a new neural network Multi-HyLSTM to overcome the programming language-specific challenges. It includes two important features, a global dependence enhancing learning module HyLSTM and a new multi-path architecture. We conducted an ablative study to compare Multi-HyLSTM with two intermediate solutions, HyLSTM, and Multi-BERT, which remove or replace one of our designs. We further compare our model with BERT and LSTM in the same task. The experiments show that our Multi-HyLSTM achieves the best accuracy of cryptographic API completion at 98.99%, showing a boost compared with BERT (92.49%) and LSTM (90.62%).

ACKNOWLEDGMENT

This work has been partly supported by the National Science Foundation under Grant No. CNS-1929701.

REFERENCES


1. Motivation

- State-of-the-art API Completion is insufficient.
- Systematical Comparisons for neural network solutions are necessary.
- Programming language specific challenges need to be identified.

2. Research Questions

What are our research questions of the comparisons?

- **RQ1:** How does program analysis guidance influence code embedding?
- **RQ2:** What are the programming language specific challenges?
- **RQ3:** How well do our neural network design choices in cryptographic API completion?

3. Program Analysis Guided Embedding Approaches

- We obtain byte code sequences, program slices, and API dependence graphs by applying program analysis.
- We train 3 types of code embeddings, byte2vec, slice2vec, and dep2vec respectively.

4. Programming Language Specific Challenges and Our Designs

- **Global dependence challenge**
  - We use a high-frequency code pattern (b, c, d) and its low-frequency variant (a, b, c, d).
  - HyLSTM gives the correct prediction for dL. When predicted wrongly, HyLSTM gives a larger loss to correct it.

- **Multi-path dependence challenge**
  - When we need to predict g1, we use
  - Single-path prediction
  - Multi-path prediction (Ours)

5. Comprehensive Comparisons

- Table 1: Accuracy comparison between different embedding settings in cryptographic API completion.
  - With program analysis, dep2vec improves the accuracy by 36.10% on average, compared with byte2vec.
  - With slice2vec, the accuracy is improved by 12.02% on average compared with its one-hot baseline.
  - With dep2vec, the accuracy is improved by 3.97% on average compared with one-hot vectors.

- Table 2: Accuracy comparison between our neural network (Multi-HyLSTM) and its intermediate baselines in cryptographic API completion. A, K, and U stand for accuracy for all cases, known cases, and unknown cases.

Ongoing Work: Publishing an API Completion Plugin and an Evaluation Benchmark

Our ongoing work is to publish an API completion plugin based on our program analysis guided embedding and neural network design. We are also preparing our cryptographic API dataset as an API completion evaluating benchmark.

### References