2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) | 978-1-6654-9598-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSE-Companion55297.2022.9793808

# Poster: Comprehensive Comparisons of Embedding Approaches for Cryptographic API Completion

Ya Xiao[1], Salman Ahmed[1], Xinyang Ge[2],Bimal Viswanath[1],Na Meng[1], Danfeng (Daphne) Yao[1]
[1]Computer Science, Virginia Tech, Blacksburg, VA
[2]Microsoft Research, Redmond, WA
{yax99,ahmedms,vbimal,nm8247,danfeng}@vt.edu,aegiryy@gmail.com

## ABSTRACT

In this paper, we conduct a measurement study to comprehensively compare the accuracy of Cryptographic API completion tasks trained with multiple API embedding options. Embedding is the process of automatically learning to represent program elements as low-dimensional vectors. Our measurement aims to uncover the impacts of applying *program analysis*, *token-level embedding*, and *sequence-level embedding* on the Cryptographic API completion accuracies. Our findings show that program analysis is necessary even under advanced embedding. The results show 36.10% accuracy improvement on average when program analysis preprocessing is applied to transfer byte code sequences into API dependence paths. The best accuracy (93.52%) is achieved on API dependence paths with embedding techniques. On the contrary, the pure data-driven approach without program analysis only achieves a low accuracy (around 57.60%), even after the powerful sequence-level embedding is applied. Although sequence-level embedding shows slight accuracy advantages (0.55% on average) over token-level embedding in our basic data split setting, it is not recommended considering its expensive training cost. A more obvious accuracy improvement (5.10%) from sequence-level embedding is observed under the cross-project learning scenario when task data is insufficient. Hence, we recommend applying sequence-level embedding for cross-project learning with limited task-specific data.

**ACM Reference Format:**
Ya Xiao[1], Salman Ahmed[1], Xinyang Ge[2],Bimal Viswanath[1],Na Meng[1], Danfeng (Daphne) Yao[1]. 2022. Poster: Comprehensive Comparisons of Embedding Approaches for Cryptographic API Completion. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3510454.3528645

## 1 INTRODUCTION

API completion aims to predict the next API method given the previous code context, which is an important building block for many software engineering tasks. When training a neural network for API completion, code embedding is a key step. Code embedding refers to the process of transforming program elements to continuous vectors [1, 4]. This transformation is important, as subsequent model training and inference are performed on the embeddings.

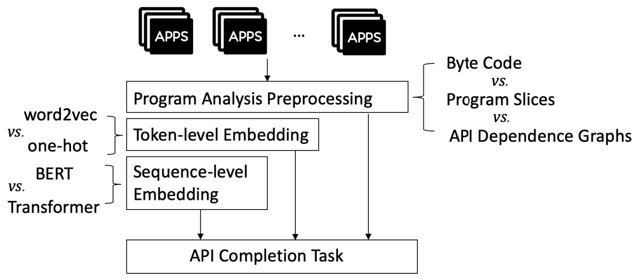

**Figure 1: Three important settings of our comparative experiments. We perform API completion tasks with different choices of program analysis preprocessing, token-level embedding, and sequence-level embedding.**

Despite existing code embedding work [1–4, 6, 9], it is still unclear their effectiveness in API completion. In this paper, we conduct comprehensive comparative experiments to uncover the impacts of multiple embedding design choices on API completion accuracy. Specifically, we compare design choices of three important decisions—*program analysis preprocessing*, *token-level embedding*, and *sequence-level embedding*, as shown in Figure 1. We choose these three dimensions because they are necessary decisions one needs to make for embeddings used in downstream task training.

We perform experiments on our Java cryptographic API benchmark collected from 79,887 Android Apps. We choose Java cryptographic APIs because the correct usage of them is complicated for developers [5, 7]. The solutions that generate accurate completion for Java cryptographic APIs would be very helpful. Our Java cryptographic API benchmark is publicly available on GitHub [1].

**Table 1: Accuracy of the cryptographic API completion with different token-level embeddings**

| LSTM | Byte Code | | Slices | | Dependence Paths | |
|---|---|---|---|---|---|---|
| Units | 1-hot | byte2vec | 1-hot | slice2vec | 1-hot | dep2vec |
| **64** | 49.78% | 48.31% | 66.39% | 78.91% | 86.00% | 86.33% |
| **128** | 53.01% | 53.52% | 68.51% | 80.57% | 84.81% | 87.75% |
| **256** | 54.91% | 54.59% | 70.35% | 82.26% | 84.57% | 91.07% |
| **512** | **55.80%** | **55.96%** | **71.78%** | **83.35%** | **86.34 %** | **92.04%** |

We explore four research questions through our experiments. **RQ1: What is the impact of program analysis preprocessing used with token-level embedding on cryptographic API completion accuracy?** To answer this question, we compare the token-level embeddings, *byte2vec*, *slice2vec*, and *dep2vec* in API completion tasks. *byte2vec* is the embedding trained on byte code sequences. *slice2vec* is the embedding on program slices while *dep2vec* is the

[1]https://github.com/Anya92929/DL-crypto-api-auto-recommendation

Table 2: Accuracy of the next API completion with or without sequence-level embedding (pretrain).

| Model Size | Byte Code | | Slices | | Dependence Paths | |
|---|---|---|---|---|---|---|
| | Trans. + byte2vec (w/o. pretrain) | byteBert (w. pretrain) | Trans. + slice2vec (w/o. pretrain) | sliceBert (w. pretrain) | Trans. + dep2vec (w/o. pretrain) | depBert (w. pretrain) |
| **Small** | 44.38% | 45.21% | 83.37% | 84.15% | 90.96% | 91.07% |
| **Base** | 56.76% | 57.59% | 84.80% | 84.83% | 92.80% | 93.52% |

embedding on API dependence paths. They differ in the program analysis preprocessing choices. We train identical LSTM models with different embedding options as inputs to achieve the cryptographic API completion. Table 1 shows the results.We observe the Finding 1 which answers our RQ1.

Finding 1: For Crypto API completion with token-level embeddings, program analysis significantly improves the accuracy by 36.10%[2] on average.

**RQ2: What is the impact of applying token-level embedding on cryptographic API completion accuracy?** To answer this question, we compared token-level embeddings with the baseline version, one-hot vectors, under identical program analysis preprocessing. Therefore, we have three groups of comparison between token-level embeddings and one-hot vectors, on byte code sequences, program slices, and API dependence paths, as shown in Table 1. We summarize our finding for RQ2 as follows.

Finding 2: For Crypto API completion on program slices and API dependence paths, token-level embedding achieves average accuracy improvement by 12.02% and 3.97%, respectively, compared with one-hot vectors.

**RQ3: What is the impact of program analysis preprocessing used with sequence-level embedding on cryptographic API completion accuracy?** We answer this question by comparing the sequence-level embeddings *byteBERT*, *sliceBERT*, and *depBERT* in cryptographic API completion. *byteBERT*, *sliceBERT*, and *depBERT* are the sequence-level embeddings trained on byte code sequences, program slices, and API dependence paths, respectively. We pretrain them on our Java cryptographic code by applying the masked language modeling on Transformer models. We observe the following finding from Table 2.

Finding 3: For Crypto API completion with sequence-level embedding, program analysis makes substantial accuracy improvement of 40.90%[3] on average.

**RQ4: What is the impact of applying sequence-level embedding on cryptographic API completion accuracy?** To show the benefit of sequence-level embeddings, we compare the cryptographic API completion with or without a sequence-level embedding. *byteBERT*, *sliceBERT*, and *depBERT* are compared with the identical but unpretrained neural network (i.e., Transformer [8]). We perform the same task-specific training for them. Table 2 suggests the following finding for RQ4.

[2]dep2vec column - byte2vec column in Table 1
[3]depBERT column - byteBERT column in Table 2

Finding 4: Despite the slight accuracy improvement (0.55% on average), sequence-level embedding is not the first recommended strategy to improve the Crypto API completion, compared with program analysis and a larger model.

**Significance of research contributions.** Our work provides the first quantitative and systematic comparison of the prediction accuracy of multiple API embedding approaches for neural network based code completion. Our rigorous experiments provide new empirical results that have not been previously reported, including how various domain-specific program analyses improve data-driven predictions. These quantitative findings, together with the new cryptographic API benchmark, help guide and design more powerful and accurate code completion solutions, leading to high quality and low vulnerability software projects in practice. In addition, our measurement methodology can be generalized to other types of APIs, beyond the specific cryptographic setting.

## ACKNOWLEDGMENT

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

## REFERENCES

[1] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29.
[2] Steven HH Ding, Benjamin CM Fung, and Philippe Charland. 2019. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 472–489.
[3] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).
[4] Jordan Henkel, Shuvendu K Lahiri, Ben Liblit, and Thomas Reps. 2018. Code vectors: Understanding programs through embedded abstracted symbolic traces. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 163–174.
[5] Na Meng, Stefan Nagy, Danfeng Yao, Wenjie Zhuang, and Gustavo Arango-Argoty. 2018. Secure coding practices in Java: Challenges and vulnerabilities. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 372–383.
[6] Trong Duc Nguyen, Anh Tuan Nguyen, Hung Dang Phan, and Tien N Nguyen. 2017. Exploring API embedding for API usages and applications. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 438–449.
[7] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng Yao. 2019. Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2455–2472.
[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.
[9] Fei Zuo, Xiaopeng Li, Patrick Young, Lannan Luo, Qiang Zeng, and Zhexin Zhang. 2019. Neural machine translation inspired binary code similarity comparison beyond function pairs.