Towards Zero Knowledge Learning for Cross Language API Mappings

Nghi D. Q. Bui
School of Information Systems, *Singapore Management University*
dqnbui.2016@phdis.smu.edu.sg

***Abstract*—Programmers often need to migrate programs from one language or platform to another in order to implement functionality, instead of rewriting the code from scratch. However, most techniques proposed to identify API mappings across languages and facilitate automated program translation require manually curated *parallel* corpora that contain already mapped API seeds or functionally-equivalent code using the APIs in two different languages so that the techniques can have an anchor to map APIs. To alleviate the need of curating parallel data and to generalize the applicability of program translation techniques, we develop a new automated approach for identifying API mappings across languages based on the idea of *unsupervised domain adaption* via Generative Adversarial Network (GAN) and an additional refinement procedure that can transform two vector spaces to align the API vectors in the two spaces without the need of manually provided anchors. We show that our approach can identify API mappings more accurately than Api2Api [25] without the need of curated parallel seeds.**

I. Introduction

API mappings across languages or platforms [26], [28], [32] can be very useful for various tasks in software engineering, e.g., software porting and reuse [2], [6], [12], [16], [17], [29].

Although there exist some studies that can construct API mappings across languages, those techniques have a common shortcoming among them that they require certain explicit knowledge about the API mappings in different languages. Either they require a large body of *parallel* program corpora that contain functionally equivalent code that use APIs in two different languages to be mapped (MAM [32] and StaMiner [20]), or they requires manual efforts to identify a large set of mapped APIs as the seeds in order to align APIs across two whole different vector spaces (Api2Api [25]), or even different form of parallel data, which is the natural language descriptions embedded in programs in different programming languages, to map APIs (DeepAM [11])

The existing approaches require a large amount of curated parallel data in the form of either code implementing the same functionality in different languages or mapping seeds used to identify some mapped code elements in different languages to bootstrap the whole mapping process. Curating such parallel data is non-trivial and often requires manual efforts and *prior knowledge* of human developers.

Our research contribution is to answer the key question: "*Can we learn API mappings across languages with (almost) zero knowledge?*" I.e., with suitable techniques, we shall be able to mine API mappings from large sets of programs in different languages (e.g., readily available from Github) without the need of manually curated parallel data.

Our intuition for resolving this question is that, given two large codebases in two languages, certain similarities in the distributions of APIs and code elements used in the two codebases may be exploited to discover API mappings across languages without manually specified parallel corpora. We realize the intuition by adapting word embedding techniques to construct vector spaces for individual languages and by adapting the Generative Adversarial Network (GAN) (e.g., [9]), which is an unsupervised domain adaptation technique, to transform vector spaces to align with each other with (almost) zero knowledge. In addition, we introduce a refinement procedure for GAN to improve the alignment between vector spaces based on various heuristics for inferring possible alignments.

II. Background and Related Work

For the problem of cross-language program translation, a.k.a. language migration, much work has utilized various statistical language models for tokens [22], phrases [14], [23], [24], or APIs [8], [20], [21], [27], [31], [32], to facilitate translation. A few studies have used word embedding, similar to our work, for API mapping and migration (e.g., [10], [11], [25], [27]), but our work does not need curated parallel corpora or mapping seeds. Tools for translating code among specific languages in practice (e.g., Java2CSharp [1]) are useful, but they are also often dependent on manually defined rules specific to grammars of individual languages.

For the techniques used to construct and transform vector spaces, we get inspirations from recent progress in NLP that use GAN to alleviate the need of parallel corpora (e.g., [4], [7], [9]), which provides the technical foundation for our work.

Formally, given two vector spaces, $X = \{x_1, x_2, ..., x_n\}$ as the source and $Y = \{y_1, y_2, ..., y_m\}$ as the target, containing n and m API embeddings for two languages, the goal is to find a linear mapping W^* such that $W^* = argmin_W ||W*X-Y||$). We can build a model according to GAN to approximate W^*, which comprises of a neural network, so-called the discriminator, trained to maximize its ability to classify any sample data from Y as belonging to Y, while minimizing its mistakes in classifying any sample data from $W * X$ as belonging to Y. On the other hand, the transformation matrix W is trained to fool the discriminator by generating samples from $W * X$ as similar to Y as possible. The overall training goal is to find W so that the discriminator is unable

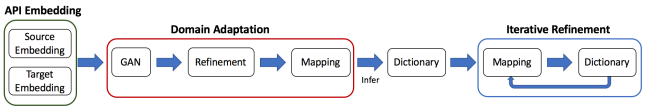

Fig. 1. Overview of our vector space alignment approach.

to accurately predict the embedding origins, i.e. find W such that $W * X$ is close to Y. We say that the vector space of X has been adapted (or, aligned) to Y via W when the training converges.

III. Approach Overview

The overview of our approach is depicted in Figure 1, which comprises of three main substeps: generate code and API embeddings, domain adaptation to align the distributions and output the mapping, and the refinement procedure to refine the mapping.

A. Code and API Embedding

We traverse through the ASTs of programs to get the code corpus, which includes APIs and important code elements (e.g programming keywords). We then use the Skip-gram model [15] to train the embeddings for the code corpus in each language to get the vector space for the code elements and APIs of the language.

B. Vector Space Alignment

First, the two vector spaces for APIs in two different languages are adapted by GAN to make them aligned with each other, so-called the domain adaptation step. The output of this step is a transformation matrix W, where W serves as the mapping between the two vector spaces, i.e., the two vector spaces are merged in one joint embedding through W.

Second, an iterative refinement procedure is applied to refine the transformation matrix. The GAN learning without parallel corpora assumes the distributions in the two vector spaces are similar, which may not be completely true for APIs, and the GAN learning may not reach optimal results. The intuition of this step is based on the assumption that the domain adaptation can, at least, produce a good initial mapping between the vector spaces. Then from the mapping, one can infer good synthetic seeds based on some heuristics. And these seeds can be used as the input for the same system in a self-learning fashion by assuming that the output mapping dictionary was indeed better than the original one, should serve to learn a better mapping and, consequently, an even better dictionary in the next iteration.

We propose auto-seeding heuristics to provide synthetic parallel mapping seeds using the W just learned with adversarial training. Specifically, we consider that (1) a simple text comparison to identify APIs having the same class and method names in two languages, (2) the top-K most frequent words, and (3) all API pairs that are "similar enough" in the joint vector space aligned by GAN, can be good seeds to use for the refinement step. The refinement step takes several iterations and validates over a testing set until satisfying the convergence criteria (i.e., the validation result no longer gets better).

TABLE I
API mapping results for Precision at 1,5,10 (P@1, P@5, P@10)

Settings	P@1	P@5	P@10
GAN	0.22	0.30	0.37
GAN + Refinement	0.34	0.53	0.63
Api2Api (no seed)	0.02	0.05	0.11
Api2Api	0.30	0.43	0.52

C. API Mapping Generation

After all API embeddings are generated and aligned, API mappings will be actually generated by near neighbor queries: Given an API in one language, the vector of the API is used to retrieve its top-k nearest neighbor vectors from the other language, and the APIs corresponding to those neighbor vectors will be reported as possible mappings of the input API.

IV. Empirical Evaluation

We collect large sets of Java and C# programs from Github, each containing approximately 2 millions of files. As the main advantage of our approach, there is no need to specify which code in Java is functionally equivalent to which code in C#.

We take 860 API mappings manually defined in Java2CSharp [1] as the ground truth for evaluating our approach. We compare our approach with Api2Api [25] as it appears to be the state-of-the-art for such a task. Since Api2APi requires seeds to train its model, we split the 860 mappings evenly into two folds: one fold is the training data for Api2APi, and the other is the testing data for both approaches. Table I compares the precisions of the mapping results of both approaches. If we do not provide seeds to Api2Api, its precisions are very low. Our approach can achieve reasonable precisions even without any knowledge. With the additional refinement procedure, our results are significantly better than Api2Api.

V. Conclusion and Future Work

This work proposes a domain adaptation approach to automatically construct and align vector spaces for code with zero knowledge. From the preliminary results, we can outperform the state-of-the-art technique for the API mapping task without the need of human effort to curate mapping seeds.

We believe our technique can be used to resolve the out-of-vocabulary (OOV) problem [3], [5], [13] for learning and modeling fast evolving software code because the embeddings of OOV words may be "adapted" and approximated on-the-fly from the known embeddings of their "contextual" words or "similar" words in different languages. Part of our future research is to integrate domain adaptation based OOV estimation into the approach for API mappings.

In addition, domain adaptation techniques may also be useful for other software engineering tasks that involve two different domains targeted by transfer learning [18], [19], [30], such as cross-language program classification, code summarization, cross-language/project bug prediction. As such, it will be interesting if two domains can be adapted with as little data as possible. In the future, we will explore more variants of GAN to solve such interesting domain adaptation problems.

REFERENCES

[1] Java2csharp. https://github.com/codejuicer/java2csharp.

[2] V. Agarwal, S. Goyal, S. Mittal, and S. Mukherjea. MobiVine: A middleware layer to handle fragmentation of platform interfaces for mobile applications. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, pages 24:1–24:10, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):81:1–81:37, July 2018.

[4] A. Conneau, G. Lample, M. Ranzato, L. Denoyer, and H. Jégou. Word translation without parallel data. *CoRR*, abs/1710.04087, 2017.

[5] M. Cvitkovic, B. Singh, and A. Anandkumar. Deep learning on code with an unbounded vocabulary. In *Machine Learning for Programming (ML4P) Workshop at Federated Logic Conference (FLoC)*, 2018.

[6] M. El-Ramly, R. Eltayeb, and H. A. Alla. An experiment in automatic conversion of legacy java programs to c#. In *IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pages 1037–1045, 2006.

[7] Y. Ganin and V. S. Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 1180–1189, 2015.

[8] A. Gokhale, V. Ganapathy, and Y. Padmanaban. Inferring likely mappings between apis. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 82–91. IEEE, 2013.

[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[10] X. Gu, H. Zhang, D. Zhang, and S. Kim. Deep API learning. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 631–642, November 13-18 2016.

[11] X. Gu, H. Zhang, D. Zhang, and S. Kim. DeepAM: Migrate APIs with multi-modal sequence to sequence learning. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3675–3681, August 19-25 2017.

[12] J.-L. Hainaut, A. Cleve, J. Henrard, and J.-M. Hick. *Migration of Legacy Information Systems*, pages 105–138. Springer Berlin Heidelberg, 2008.

[13] V. J. Hellendoorn and P. Devanbu. Are deep neural networks the best choice for modeling source code? In *11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 763–773, New York, NY, USA, 2017. ACM.

[14] 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*, Onward! 2014, pages 173–184, New York, NY, USA, 2014. ACM.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[16] J. D. Mooney. Portability and reusability: Common issues and differences. In *ACM 23rd Annual Conference on Computer Science*, CSC '95, pages 150–156, 1995.

[17] M. Mozumdar, F. Gregoretti, L. Lavagno, and L. Vanzago. Porting application between wireless sensor network software platforms: Tinyos, mantis and zigbee. pages 1145–1148, 09 2008.

[18] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan. Heterogeneous defect prediction. *IEEE Transactions on Software Engineering*, 44(9):874–896, Sep. 2018.

[19] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 382–391, 2013.

[20] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen. Statistical learning approach for mining API usage mappings for code migration. In *ACM/IEEE International Conference on Automated Software Engineering (ASE)*, pages 457–468, 2014.

[21] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen. Statistical learning of API mappings for language migration. In *36th International Conference on Software Engineering - Companion (ICSE)*, pages 618–619, May 31 - June 07 2014.

[22] A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen. Lexical statistical machine translation for language migration. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 651–654, August 18-26 2013.

[23] A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen. Divide-and-conquer approach for multi-phase statistical migration for source code (T). In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 585–596, November 9-13 2015.

[24] A. T. Nguyen, Z. Tu, and T. N. Nguyen. Do contexts help in phrase-based, statistical source code migration? In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 155–165, October 2-7 2016.

[25] T. D. Nguyen, A. T. Nguyen, H. D. Phan, and T. N. Nguyen. Exploring API embedding for API usages and applications. In *39th International Conference on Software Engineering (ICSE)*, pages 438–449, 2017.

[26] R. Pandita, R. Jetley, S. D. Sudarsan, T. Menzies, and L. Williams. TMAP: discovering relevant API methods through text mining of API documentation. *Journal of Software: Evolution and Process*, 29(12), 2017.

[27] H. D. Phan, A. T. Nguyen, T. D. Nguyen, and T. N. Nguyen. Statistical migration of API usages. In *39th International Conference on Software Engineering - Companion Volume (ICSE)*, pages 47–50, May 20-28 2017.

[28] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford. Automated api property inference techniques. *IEEE Trans. Softw. Eng.*, 39(5):613–637, May 2013.

[29] M. Shoaib, A. Ishaq, M. A. Ahmad, S. Talib, G. Mustafa, and A. Ahmed. Software migration frameworks for software system solutions: A systematic literature review. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 8(11):192–204, 2017.

[30] S. Yan, B. Shen, W. Mo, and N. Li. Transfer learning for cross-platform software crowdsourcing recommendation. In *24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 269–278, Dec 2017.

[31] H. Zhong, S. Thummalapenta, and T. Xie. Exposing behavioral differences in cross-language API mapping relations. In *Proceedings of 16th International Conference on Fundamental Approaches to Software Engineering (FASE), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS)*, pages 130–145, March 16-24 2013.

[32] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang. Mining API mapping for language migration. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE)*, pages 195–204, May 1-8 2010.