Providing a labeled statements dataset to enhance the trans-compilation-based tools

Abdelrahman M. Soliman1, Moataz-Bellah.A.Radwan1, Hussain.E.Eida,1

Amira.T.Mahmoud1,2, Youmna.E.Mohammed1,2, Walaa.M.Asal1,2,4, Ahmed H. Yousef1,2,3

[Abd.Soliman@nu.edu.eg](mailto:Abd.Soliman@nu.edu.eg) , [Mo.Radwan@nu.edu.eg](mailto:Mo.Radwan@nu.edu.eg) , [Hu.elshazly@nu.edu.eg](mailto:Hu.elshazly@nu.edu.eg) , [AmTarek@nu.edu.eg](mailto:AmTarek@nu.edu.eg), [yeid@nu.edu.eg](mailto:yeid@nu.edu.eg), [WMedhat@nu.edu.eg](mailto:WMedhat@nu.edu.eg), [ahassan@nu.edu.eg](mailto:ahassan@nu.edu.eg)

*1 School of Information Technology and Computer Science Nile University Giza, Egypt*

*2 Center for Informatics Science, Nile University, Giza*

*3 Faculty of Engineering, Ain Shams University, Cairo*

*4 Faculty of Computer and AI, Banha University, Banha*

***Abstract*— Nowadays Mobile Applications are a necessity as everyone is depending on them in their everyday tasks. We use them for communication, entertainment, and utilities. Every day new devices are introduced to the market. The diversity in these devices resulted in many platforms like Android and iOS. These different mobile platforms used by different companies and manufacturers made it challenging for mobile developers to build an application or a system that can be deployed on different platforms. Developers will be required to go through multiple development cycles to develop the application for each of the targeted platforms. To solve these issues, cross-platform frameworks emerged as a solution that would help the developers to write one codebase and deploy it on multiple platforms at the same time. Even though cross-platform frameworks reduced the time and cost needed to develop an application, but they did not provide the native feel and performance of the native applications. It was also challenging for developers to integrate cross-platform applications with a pre-existing system and to maintain the system. Some approaches were introduced to solve these issues by allowing the developer to write one codebase and generate the native code for each platform. One of the recently presented approaches is the trans-compilation approach that depends on the parser code to recognize the source code and generate the corresponding code for it in the target language. Even though the trans-compiler- based approaches were effective in providing an application that combines the advantages of the cross-platform and native development, it faced some problems in dealing with statements that is not expressed in the language’s grammar rules, eternal APIs, and APIs that require code definition, like Firebase. In this paper, our aim is to provide a solution that can be integrated with the trans-compilation approach to solve the previously mentioned issues to enhance the conversion success rate by recognizing statements and predict the corresponding ones**

***Keywords—cross-platform, code conversion, code generation mobile applications, trans-compilation, data mining, machine learning.***

1. INTRODUCTION

The existence of a lot of mobile devices and manufacturers resulted in many platforms, each platform requires the mobile application to be written in platform specific language. Developers have to go through multiple development cycles, one for each platform. This process costs companies and developers a lot of money, time, and manpower to cover the target platforms.

To fix this issue, cross-platform frameworks emerged as a solution to help reduce the time and cost needed. Cross- platform frameworks allow developers to write code once and deploy it in multiple platforms at the same time. Many of these cross-platform frameworks are used commercially like

Flutter [1], React Native [2], Xamarin [3], Ionic [4], and

Cordova [5].

Even though cross-platform frameworks solved some of the development problems by reducing the time and cost needed to cover multiple platforms, it did not produce an application that can provide the native feel and performance [6][7][16][17][18][19][20][21] as illustrated in these studies. To overcome this challenge, some approaches were introduced to help developers write code once and run it on multiple platforms while maintaining the native feel and performance. One of these approaches was the trans-compilation-based code conversion approach introduced in [10][11][12]. The trans-compilation approach is all about using the grammar file to generate the parse code and tree using ANTLR [13] (Another Tool for Language Recognition). ANTLR is a tool that gets the language’s grammar file as input containing the statements rules and generate the parse code and parse tree for any given code statements in that language. The parse code is an abstract class that contains function definitions representing each statement in the source language. In [10][11][12], developers proceeded their conversion process by overriding the parse code of the source language to generate the visitor class which acts as a tree walker. The visitor class is used to recognize the inserted source language statements and then extract the desired tokens necessary for the conversion process. After that, the extracted tokens are used and casted in a pre-written code templates to complete the conversion process.

While using the Trans-compilation approach, some issues become noticeable in the output code. There are some obstacles in the conversion process that trans-compilation cannot deal with like working with built-in functions and pre- defined APIs, like firebase API for example. These issues occur due to the nature of the programming language’s grammar rules. The previously mentioned examples are all accepted by the grammar of the language, but they are being dealt with as normal statements without any specification. The grammar deals with built-in functions call as a normal call, and the APIs that needs definition (like firebase) as normal code block leading to inaccurate code generation. In.

[23] the authors used neural based code conversion to convert mobile applications code from one native language to another. However, due to the lack of corresponding code statements in different programming languages like swift and java, they had to make the data set from scratch. This made the neural-based conversion genericness and coverage for all backend and UI statements nearly impossible. Our goal is to enhance the trans-compilation-based tools with machine learning model that will be assigned to deal with the previously mentioned issues that the trans-compilation couldn’t cover. The model will discard the language grammar rules and convert the statements in the needed way. In the phase of this paper, we are mainly focusing on the data collection phase of our project. To overcome the dataset problem, we plan to build visitor classes for each of our source and destination languages and combine them to build statements collector that does the labeling process in parallel. When given an expression, the visitor class will parse it with a specific function for each statement rule. Then, each statement is labeled and returned to the dataset using its specific visiting function. The Aim of our study is to introduce a solution that combines all the upsides of the approaches in [10][11][12][14][15] and introduce the following: 1) Build a huge dataset to provide enough training data for the models. 2)Enhancing the approaches introduced in [10][11][12] using some machine learning techniques. And to illustrate our methodology, we are going to implement code miners for Xamarin, which is written in C#, as well Java for Android and Swift for iOS.

II. METHODOLGY

The aim of this research is to build a dataset of labeled statements of each of our chosen languages (Swift, Java, and C#) to be able to use it to train a model to recognize the rule of a given statement. Therefore, the methodology consists of two phases, data collection phase and prediction phase.

1. *Data Collection*

In order to collect our dataset, we decided to use ANTLR tool which was used in [8][9][11][14][15], they used ANTLR [13] to build a visitor class that allow them to parse a given source code. The way ANTLR work is that it takes an input grammar file and generate a parse code and a parse tree. Fig1. Shows ANTLR workflow.

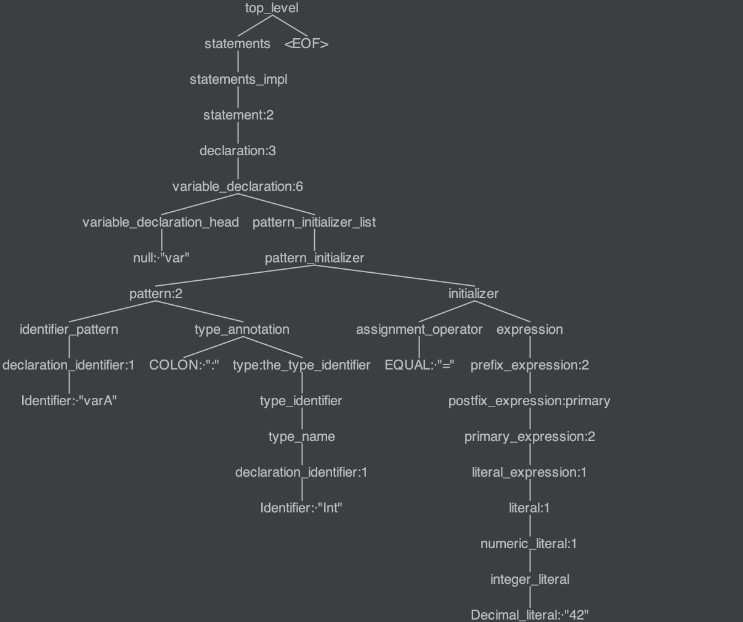
Diagram

Description automatically generated

*Figure* 1*: ANTLR Workflow*

The parse code we generate is a class that contains a visitor function for each statement in the language’s grammar.

This parse code is overridden to build a visitor class which is used to parse statements. To be able to collect our dataset, we are going to build miners using these visitor classes for each language. To collect statements and label them, we will use the visiting functions to be able to exactly match the statement with its label. Unlike the work done for the conversion processes in [8][9][11][14][15], we will not use



*Figure 2: Parse Tree*

Compared to the visiting procedure in the previously mentioned approach, we will follow the same visiting flow but without intervening in all the visiting functions, instead we will visit the most comprehensive statement in the branch. So as an example of labeling and returning the variable declaration statement is by returning the whole context of (variable\_declaration) object on Figure2. We do the same for each of the target statements. Each time a statement is recognized, we return the context of the whole object alongside it’s label to a csv file to construct the dataset as shown in table 1.

*Table 1 Statements Sample*

|  |  |
| --- | --- |
| **Statement** | Label |
| **var window:UIWindow?** | variable\_declaration |
| **func applicationWillResignActive (\_application:UIApplication) { }** | function\_declaration |
| **func applicationDidEnterBackground (\_application:UIApplication) { }** | function\_declaration |
| **@UIApplicationMain AppDelegate { }** | class\_statement |

1. *Prediction Phase*

In machine learning-based programming languages conversion, previous approaches [10][16][22] used to work with symmetrically matched datasets of statements and their exact match on the destination language which is hard to collect as it must be built by hand to generate the exact match. Some other approaches used to recognize the statement by matching code snippets with their code documentations like [11] where they managed to match a library with its correspondents by recognizing its

functionalities and match it with another correspondent with the same functionalities.

For our chosen approach we are planning to use word 2 vector approach. The nature of the statements as a text makes it hard to classify them. Therefore, in this approach we decided to convert the statements to a vector of words and encode them, as working with numbers is much easier and will be more accurate. In addition, there are repetitive patterns in particular statements. For example, in functions in swift, we have (func, (, ), ->, {, and }). These patterns will make it easier to reduce the differences between functions declarations, like how vectors in the cosine similarity algorithms work.

III.RESULTS AND DISCUSSION

To validate our approach, we decided to test the accuracy and the precision of our model with each Rule in the grammar of the three languages to check the accuracy of the model. Then we measure the actual accuracy of the output which is the precision for the three languages (Swift, Java, and C#). Tables 2, 3 and 4 show the accuracy and precision for predicting statements of swift, java and C# respectively.

1. *Swift*

For swift we can notice a high accuracy in variable declaration and function definition as it has a different characteristics than the other statements like the keyword (func) in the function definition and (Var and “:” ) in variable declaration.

*Table 2 Swift Accuracy*

|  |  |  |
| --- | --- | --- |
| **Swift** | **Accuracy** | **Precision** |
| **Variable Decl** | 95% | 90% |
| **Function Decl** | 99% | 99% |
| **Function Call** | 84% | 79% |
| **Class Decl** | 92% | 89% |
| **If Conditions** | 92% | 88% |
| **Loops** | 87% | 83% |
| **Average** | 91% | 88% |

As we can see in table 2, we can also find that we get less accuracy in function calls as it does not contain the same tokens that can give function calls its own characteristics as the main token in the function call is the name of the function which is not always a keyword.

1. *Java*

In Java we can find a high accuracy in Class Declarations due to the nature of the language, as it is an Object-Oriented Programming (OOP) language. Therefore, almost all of the files passed to miners in the data collection phase were contained in classes which increased the amount of class declaration statements in our dataset which led to a higher accuracy in testing.

*Table 3 Java Accuracy*

|  |  |  |
| --- | --- | --- |
| **Java** | **Accuracy** | **Precision** |
| **Variable Decl** | 91% | 83% |
| **Function Decl** | 93% | 89% |
| **Function Call** | 89% | 77% |
| **Class Decl** | 95% | 93% |
| **If Conditions** | 94% | 94% |
| **Loops** | 92% | 87% |
| **Average** | 92% | 87% |

A lower accuracy in variable and function declarations was recorded because both statements contain the same set of tokens like (public, private, static, and the datatypes) which created a confusion for the model in differentiating between functions and variables

1. *C#*

For the results of C#, we can see that even though the accuracy of function declarations’ accuracy is high, the precision is still low for the same reason mentioned in the previous section. As the model cannot easily differentiate between functions declaration and variable declarations due to the similarity between the keywords used in both statements

*Table 4 C# Accuracy*

|  |  |  |
| --- | --- | --- |
| **C#** | **Accuracy** | **Precision** |
| **Variable Decl** | 92% | 85% |
| **Function Decl** | 96% | 87% |
| **Function Call** | 90% | 82% |
| **Class Decl** | 92% | 89% |
| **If Conditions** | 95% | 90% |
| **Loops** | 90% | 86% |
| **Average** | 92% | 86% |

For the function call, we can notice a higher accuracy from table 4 because of the nature of the language and the number of objects used in the provided source codes.

In the flow control statements we can justify the increase in the accuracy by the number of keywords that can differentiate the statements and help the model to predict its label better.

IV. CONCLUSION AND FUTURE WORK

Mobile phones are a necessity these days and we use it in our daily tasks and for entertainment. Due to its widespread, there are many platforms with its own collection of mobile applications. The huge number of users and the diversity in platforms made it challenging for developers to cover the market’s needs. Cross-platform frameworks were a good choice helping in reducing the time and cost needed for the development. Later on, the trans-compilation approach was introduced as a solution embracing the code once and deploy on multiple platforms concept, while maintaining the native feel and performance. In this research, we worked on enhancing the trans-compilation-based code conversion approaches by providing a dataset of labeled statements that can be used to detect and convert irregular statements, like APIs and external libraries. For the future work, we are willing to collect more data covering a wider range of languages and frameworks to help producing an approach

that integrates machine learning with trans-compilation to enhance the conversion process.

1. REFERENCES
2. Flutter - Beautiful native apps in record time. (n.d.). Retrieved June 21, 2020, from https://flutter.dev/?gclid=Cj0KCQjwirz3BRD\_ARI sAImf7 LPI3GO6NQFGZttLHwT7X\_pc EF1AnovM6nJzDRRiiqSzF2sZBkTTLwaAtOrEA Lw\_wcB&gcl src=aw.ds
3. React Native · A framework for building native apps using React. (n.d.). Retrieved June 21, 2020, from https://reactnative.dev/
4. Xamarin | Open-source mobile app platform for

.NET. (n.d.). Retrieved March 25, 2020, from https://dotnet.microsoft.com/apps/xamarin

1. Ionic Website Documentation <http://ionicframework.com/docs/>
2. Documentation - Apache Cordova. (n.d.). Retrieved March 25, 2020, from https://cordova.apache.org/docs/en/latest
3. Luis Corral, Andrea Janes, Tadas Remencius, Potential Advantages and Disadvantages of Multiplatform Development Frameworks–A Vision on Mobile Environments, Procedia Computer Science, Volume 10, 2012, Pages 1202-1207, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2012.06.173.
4. Xanthopoulos, Spyros; Xinogalos, Stelios

(2013). *[ACM Press the 6th Balkan Conference in Informatics - Thessaloniki, Greece (2013.09.19- 2013.09.21)] Proceedings of the 6th Balkan Conference in Informatics on - BCI '13 - A comparative analysis of cross-platform development approaches for mobile applications. , (), 213–.* doi:10.1145/2490257.2490292

1. A. Ebone, Y. Tan and X. Jia, "A Performance Evaluation of Cross-Platform Mobile Application Development Approaches," *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2018, pp. 92-93.
2. W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, “Taxonomy of Cross-Platform Mobile Applications Development Approaches,” Ain Shams Engineering Journal, vol. 8, no. 2. pp. 163–190, 2017 [Online]
3. R. B. Hamza, D. I. Salama, M. I. Kamel and A. H. Yousef, "TCAIOSC: Application Code Conversion," *2019 Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, 2019, pp. 230-234, doi: 10.1109/NILES.2019.8909207.
4. A. A. Muhammad, A. T. Mahmoud, S. S. Elkalyouby, R. B. Hamza, and A. H. Yousef, “Trans-Compiler based Mobile Applications code converter: swift to java,” in 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), Oct. 2020, pp. 247–252, doi: 10.1109/NILES50944.2020.9257928.
5. A. T. Mahmoud, M. A. Radwan, W. Medhat, A. H. Youssef,”Crossplatform to Native Trans Compiler Based Code Conversion, Information Technology and Computer Science School, Nile University. manuscript submitted for publication.
6. T. Parr, The Definitive ANTLR 4 Reference.

Pragmatic Bookshelf, 2013

1. A. A. Muhammad, A. M. Soliman, S. Selim and A.

H. Yousef, "Generic Library Mapping Approach for Trans-Compilation," *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, 2021, pp. 62-68, doi: 10.1109/MIUCC52538.2021.9447641.

1. A. A. Muhammad, A. M. Soliman, S. Selim and A.

H. Yousef, "Automated Library Mapping Approach for Trans-Compilation,” , Information Technology and Computer Science School, Nile University. manuscript submitted for publication.

1. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: a Method for Automatic Evaluation of Machine Translation.”
2. W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, “Taxonomy of Cross-Platform Mobile Applications Development Approaches,” *Ain Shams Eng. J.*, vol. 8, no. 2, pp. 163–190, Jun. 2017, doi: 10.1016/J.ASEJ.2015.08.004.
3. A. Biørn-Hansen, T.-M. Grønli, and G. Ghinea, “A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development,” *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1–34, Nov. 2018, doi: 10.1145/3241739.
4. A. Ribeiro and A. R. Da Silva, “Survey on cross- platforms and languages for mobile apps,” in *Proceedings - 2012 8th International Conference on the Quality of Information and Communications Technology, QUATIC 2012*, 2012, pp. 255–260, doi: 10.1109/QUATIC.2012.56.
5. M. Latif, Y. Lakhrissi, E. H. Nfaoui, and N. Es- Sbai, “Cross platform approach for mobile application development: A survey,” in *2016 International Conference on Information Technology for Organizations Development (IT4OD)*, 2016, pp. 1–5, doi: 10.1109/IT4OD.2016.7479278.
6. S. Charkaoui, Z. Adraoui, and E. H. Benlahmar, “Cross-platform mobile development approaches,” in *Colloquium in Information Science and Technology, CIST*, 2015, vol. 2015-Janua, no. January, pp. 188–191, doi: 10.1109/CIST.2014.7016616.
7. TildeMT, https://[www.letsmt.eu/Bleu.aspx(accessed](http://www.letsmt.eu/Bleu.aspx(accessed) Sep.28,2020)
8. M. H. Hassan, O. A. Mahmoud, O. I. Mohammed,

A. Y. Baraka, A. T. Mahmoud and A. H. Yousef, "Neural Machine Based Mobile Applications Code Translation," 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), 2020, pp. 302-307, doi: 10.1109/NILES50944.2020.9257935.