Paper Summary: CODE GENERATION FOR MOBILE LANGUAGES USING LARGE LANGUAGE MODELS

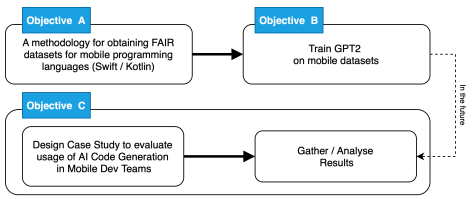
Generare De Cod Pentru Limbaje Mobile Utilizând Modele De Limbaj

Mircea-Serban Vasiliniuc

October 18, 2023

1 Objectives

The paper will present the current state of AI-Assisted Software development, focusing on the Code Intelligence concept together with associated tasks, dedicated models, datasets, and benchmarks.



It will exploit the lack of support for existing datasets for mobile languages and present the methodology we used to create public datasets for Swift and Kotlin languages that are published following FAIR principles.

We go thru the process of training a GPT-2 model with the resulting datasets in an attempt to provide code generation specific to mobile development languages.

In the final chapter, we present an extensive case study involving 16 participants from a software development department designed to understand the impact of using LLM trained for code generation in specific phases of the team, more specifically, technical onboarding and technical stack switch.

2 Creating a dataset, for mobile code, using FAIR principles

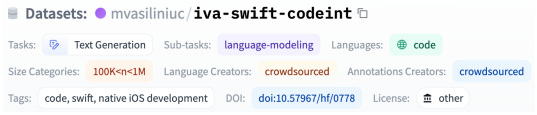
In this paper, we use an unlabeled part of a dataset to train GPT-2, a large language model to adapt it for code generation in mobile-specific languages like Swift or Kotlin. For this task, a large corpus with Swift and Kotlin code can be useful without being accompanied by labels. In the following section, we will present a methodology to create datasets for programming languages of less wide circulation, using Kotlin and Swift as examples.

Data Gathering This methodology uses the terabytes of code repositories that are open for usage and download on GitHub. For each mobile language, Swift and Kotlin we created a dataset and a table in the Google Big Query Project, Created a bucket in Google Cloud Storage, Created a query in Google Big Query Project, and Exported the resulting dataset into the bucket created. Extraction of Swift files resulted in 2.7 TB Processed, number of extracted rows/files was 753,693, total logical bytes was 3.05 GB.

Dataset Curating GitHub repositories have varying quality levels because anyone can make them. To achieve good results in real-world situations, some choices need to be made carefully. Adding some noise to the training data can help the system deal with noisy inputs better during inference. Mainly the curating rules can be summarized in three major categories: (a) removal of duplication files based on file hash, near duplication based MinHash and Jaccard similarity (b), removal of unwanted files (templates, autogenerated, unit-tests, configurations, without operators with a low rate of alphanumeric chars and (c) ratio of tokens generated after initial tokenization.

Dataset Publishing - Following FAIR Principles and HuggingFace Dataset Cards Rules FAIR is a set of core principles for effective data management and stewardship that was proposed in 2016. They aim to make data findable, accessible, interoperable, and reusable (FAIR) so that they can be used for scholarly purposes.

Figure 1: IVA-Swift-CodeInt Dataset Tags



The four main datasets published with this paper achieved a score of 87% FAIR score in DANS evaluation (77% at Findable, 100% at Accessibility, 100% at Interoperability, 72% at Reusability). Similar results were obtained using the Australian Research Data Commons tool.

Resulted datasets are publicly available on the HuggingFace platform respecting the platform publication standards (Dataset cards) and include DOI. The information contained in the dataset publication includes Dataset Description (Dataset Summary, Supported Tasks and Leaderboards, and Languages), Dataset Structure (Data Instance, Data Fields, Data Splits), Dataset Creation (Should contain Curation Rationale and Source Data) and Considerations for Using the Data.

3 Building a Model for Mobile Languages

The approach in this paper aims to train a large model, GPT-2 (small version - 239.4M parameters) from scratch to obtain results in the text-to-code task tailored for niche programming languages used in native mobile development. More precisely, we build one model for each programming language, Swift and Kotlin. In our case, the datasets are large and unlabeled. As the quantity of training data you have access to approaches the amount of data used for pretraining, it becomes an argument in favor of considering training the model and the tokenizer from scratch.

The steps in our approach involve the following and are elaborated on in the paper (a) Creating datasets for training tokenizer and model (for Swift and Kotlin), (b) Training the standard open-source GTP-2 Tokenizer, (c) Preparing datasets for processing (d) Initializing and Training the standard open-source GPT-2 Model (small version).

Since this is a causal language model, it is evaluated by calculating the cross-entropy Loss and perplexity. To ensure a standard level of reproductivity all information, approach, code, datasets, and results are publicly available.

The computation environment used for the study was provided by the Technical University of Cluj Napoca.

The model is capable of successful code generations, especially for prompts related to the usage of Platform API which can be of increased usage with a developer adapting to a new platform. Some examples of mobile-specific code, we see the output of the Swift-trained model performing code generation for a task that is clearly platform-specific, like retrieving the current time zone of the device. We can see functional code, correct usage of API, static call of current calendar on NSCalendar (NSFoundation framework from iOS SDK), and correct function call.

```
/*
A function that returns the time zone currently set on the device.
*/
public func getTimezone() -> NSTimeZone {
  let calendar = NSCalendar.currentCalendar()
  let timeZone = calendar.timeZone()
  return timeZone
}
```

```
/*
Function that saves a given String in UserPreference
*/
public func save(_ text: String) {
  let defaults = UserDefaults(suiteName: kSuiteName)
  defaults?.set(text, forKey: kAppIdKey)
  defaults?.synchronize()
```

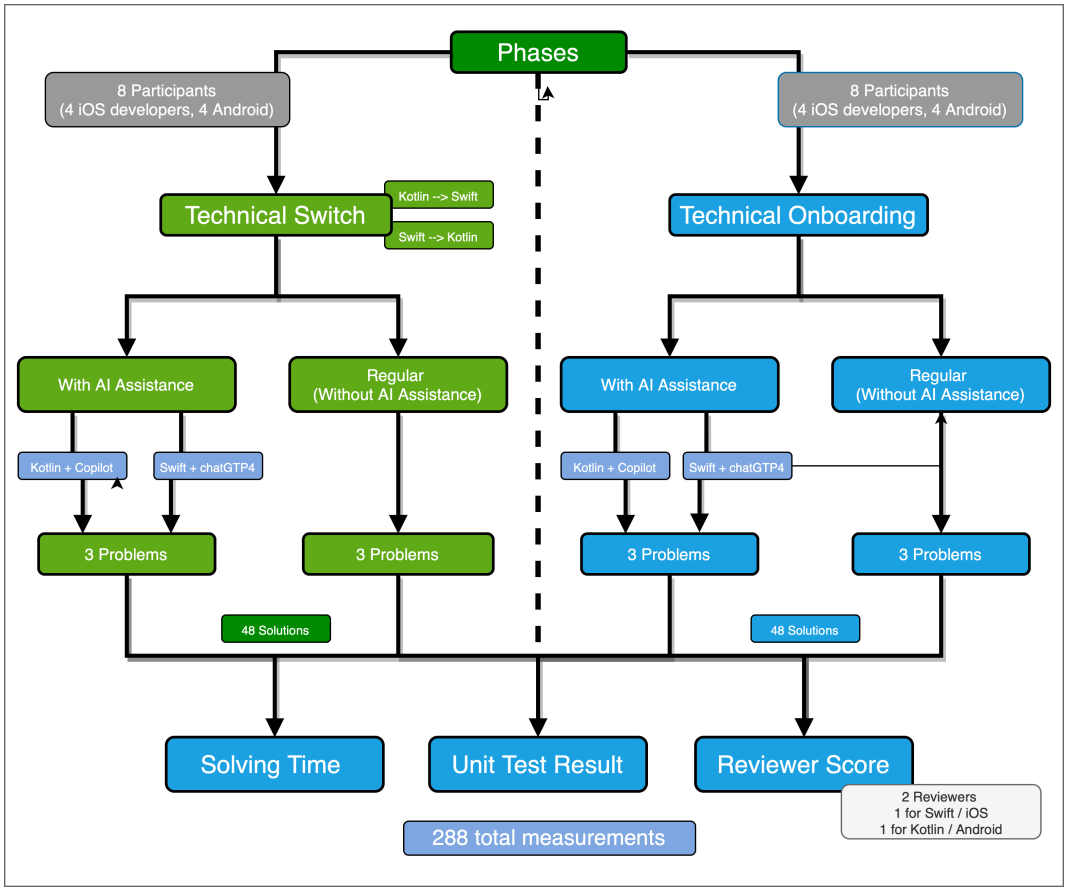
Figure 2: The code in black color is the prompt, and the code in blue is the generated code.

4 Case Study: Using AI-Assisted Code Generation

In order to analyze the impact of AI-assisted programming in a native mobile development environment, the study was designed to compare the output of development problems with and without the assistance

of AI systems for each phase: technical onboarding, and technical stack switch. The aim is to cover both Quantitative and Qualitative metrics using Reviewer's analysis together with a survey that was filled out by the participants after the tasks.

Figure 3: Overview of the Study Design.



The study aimed to answer the following research questions:

RQ1. How can an AI-based code generator affect the experience when onboarding a new team member or switching technical stacks of an existing colleague?

- For these particular types of procedures in a mobile development team, meaning providing technical assistance for a new member or assisting a colleague to adopt the sibling technical stack, we can assess that AI-Based code generators can improve the experience by providing an additional source of information which if combined with developer experience and analytic thinking can lead to rewarding results for both the subject of the procedure and the mentors assisting the procedure.

RQ2. Can AI-based code generators affect the performance (completion time, correctness) of technical onboarding or technical stack switch tasks?

- Results of the study reflect a clear improvement in the duration of achieving tasks, in a simulation of the two types of procedures, without major implications related to the correctness of the solutions provided. This can be seen as a positive effect on performance and it can be considered reasonable for mobile teams to adopt such solutions with the consideration that the long-term learning and information retention of the members must be measured before scaling the process to a standard one.

RQ3. Can AI-based code generators affect the technical integration efforts of a mobile development team?

– The design of the study considers the notable factor of technical alignment with the requirements of a team, by the addition of the ReviewerScore, in order to measure the impact of integration efforts. Results show that the impact of using such tools can lower the alignment due to reliance on a tool and less on the team's context and requirement, still, the risk can be taken into consideration when designing procedures for technical onboarding or technical task switch.

Effective prompt engineering is an attribute that was again observed to affect the results and understanding of the problem and its solution in a significant manner. We must note that candidates that were more attentive at modifying the prompts used in communication with the model have higher scores in correctness and technical alignment (ReviewerScore).

The Post Tasks Survey conveys rating the usage of the AI-Assisted tool in the following categories: overall helpfulness, level of understanding of generated code, confidence in the code, and future usage. The 16 developers, were invited to provide additional feedback as part of the survey. This is available in the appendix section of the paper.