CODE GENERATION FOR MOBILE LANGUAGES USING LARGE LANGUAGE MODELS

Generare de cod pentru limbaje mobile utilizand modele de limbaj

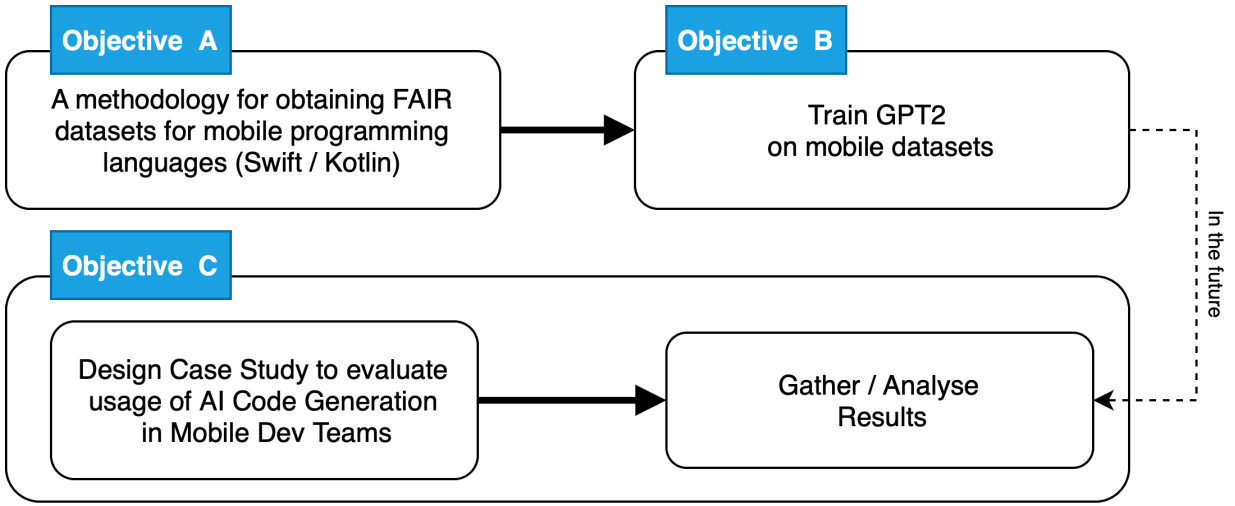
Mircea-Serban Vasiliniuc[1]

[1]Faculty of Automation and Computer Science
Technical University of Cluj-Napoca

October 18, 2023

Objectives

1. **Creating a dataset, for mobile code, using the FAIR principles**
   1. Data Gathering
   2. Data Curating
   3. Data Publishing (FAIR)
2. **Building a Model for Mobile Languages**
   1. Training GPT-2 on Mobile Datasets
   2. Results
3. **Case Study: Using AI-Assisted Code Generation**
   1. Usage of Code Generation in Mobile Team Procedures
   2. Quantitative and Qualitative Results



Methodology for creating a dataset - Data Gathering

Google Big Query, GitHub Dump, SQL query to extract the raw Swift / Kotlin source files

Extraction of Swift files resulted in

- 2.7 TB Processed
- number of extracted rows/files was 753,693
- total logical bytes 3.05 GB
- The result amounts to 3 json.gz files in a total of 712 MB.

Table 3.1: Structure of the constructed dataset.

Field	Type	Description
repository	string	name of the GitHub repository
path	string	path of the file in GitHub repository
copies	string	number of occurrences in dataset
code	string	content of source file
size	string	size of the source file in bytes
license	string	license of GitHub repository

The structure of the dataset gathered via GitHub 'dump' using Google Big Query.

Dataset Curation

Creating the 'clean' dataset - 10 rules of curating. Some curating rules:

1. Duplication (Near) with file hash, MinHash, and Jaccard similarity
2. Non-code files: templates, unit tests, generated, configs, etc.
3. Content-based (Rate of alphanumeric characters less than 0.3 of the file, line limits, file limits, '=' appearance, the ratio of language keywords, etc.)
4. Ration between chars before and after tokenization

Table 3.2: Size Differences: Raw vs. Curated

Metric	Swift Raw	Swift Curated	Kotlin Raw	Kotlin Curated
Number of total files	753693	383380 - 49 %	464215	201843 - 56 %
Number of files below 500 Bytes	129827	3680 - 97 %	99845	3697 - 96 %
Avg. Content Size (Bytes)	4245	5942 + 40 %	3252	5205 + 59 %

A simplistic comparison of the elimination process looking into different quantitative characteristics.

Dataset Publication (FAIR)

Multiple datasets were published for each programming language.

1. Two raw datasets published under FAIR principles
2. Dataset cards with summary, supported tasks, languages, data instances, fields, splits, curation rationale, source, considerations for using the data, research identifiers, and author details.
3. Training and Validation splits available
4. Tokenised versions available for reduced training times

Listing 3.2: DOI citation for dataset containing Raw Swift language

```
@misc {mircea_vasiliniuc_2023,
    author          = { {Mircea Vasiliniuc} },
    title           = { iva-swift-codeint (Revision c09ebf8) },
    year            = 2023,
    url             = {
    https://huggingface.co/datasets/mvasiliniuc/iva-swift-codeint
    },
    doi             = { 10.57967/hf/0778 },
    publisher       = { Hugging Face }
}
```

Training GPT-2 model - Steps

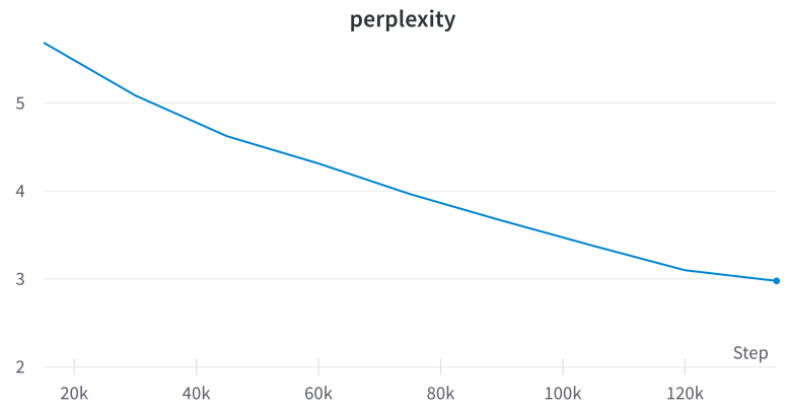
1. GPT2-Tokenizer (BPE), training it on Swift / Kotlin datasets obtained earlier.
2. Training GPT-2 (small) Model. 239.4M parameters. 960 MB.
3. Evaluation: Perplexity and Evaluation Loss.

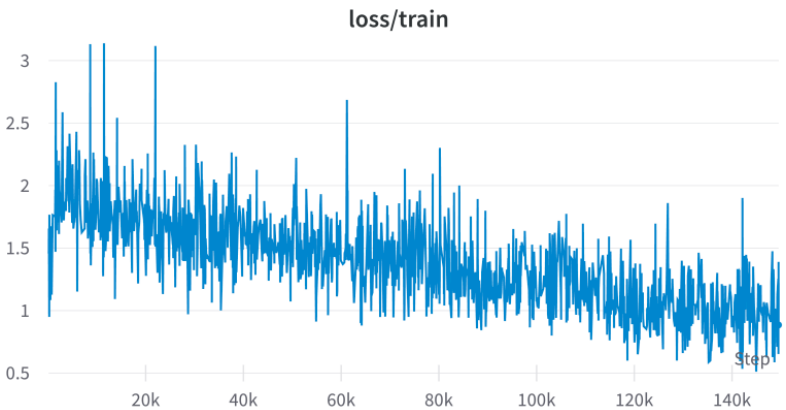
seq length	1024
weight decay	0.1
learning rate	0.0005
shuffle buffer	10000
max train steps	150000
mixed precision	fp16

num warmup steps	2000
train batch size	5
valid batch size	5
lr scheduler type	cosine
gradient checkpointing	FALSE
gradient accumulation steps	1


```
"loss/train": 0.9347587823867798,
"time_per_iteration": 0.3986995220184326
"_step": 149999,
"_runtime": 78899.45855522156,
"tflops": 16.874756165293206,
"samples": 749995,
"loss/eval": 1.0912784337997437,
"_timestamp": 1687694268.8288612,
"perplexity": 2.978078842163086,
"loss_per_step/train": 0.934758722782135
"lr": 2.2529228482781605e-13,
"steps": 149998
```


Training GPT-2 model - Results






```
/*
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()
```

```
/*
Function that posts a NSNotification named "MySpecial"
*/
public func post(userInfo: NSDictionary, value: AnyObject?, callback: () -> Void) {
  NSNotificationCenter.defaultCenter().postNotificationName("MySpecial", object:
nil, userInfo: userInfo)
```

```
/*
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
}
```

Case Study: Using AI-Assisted Code Generation - Goals

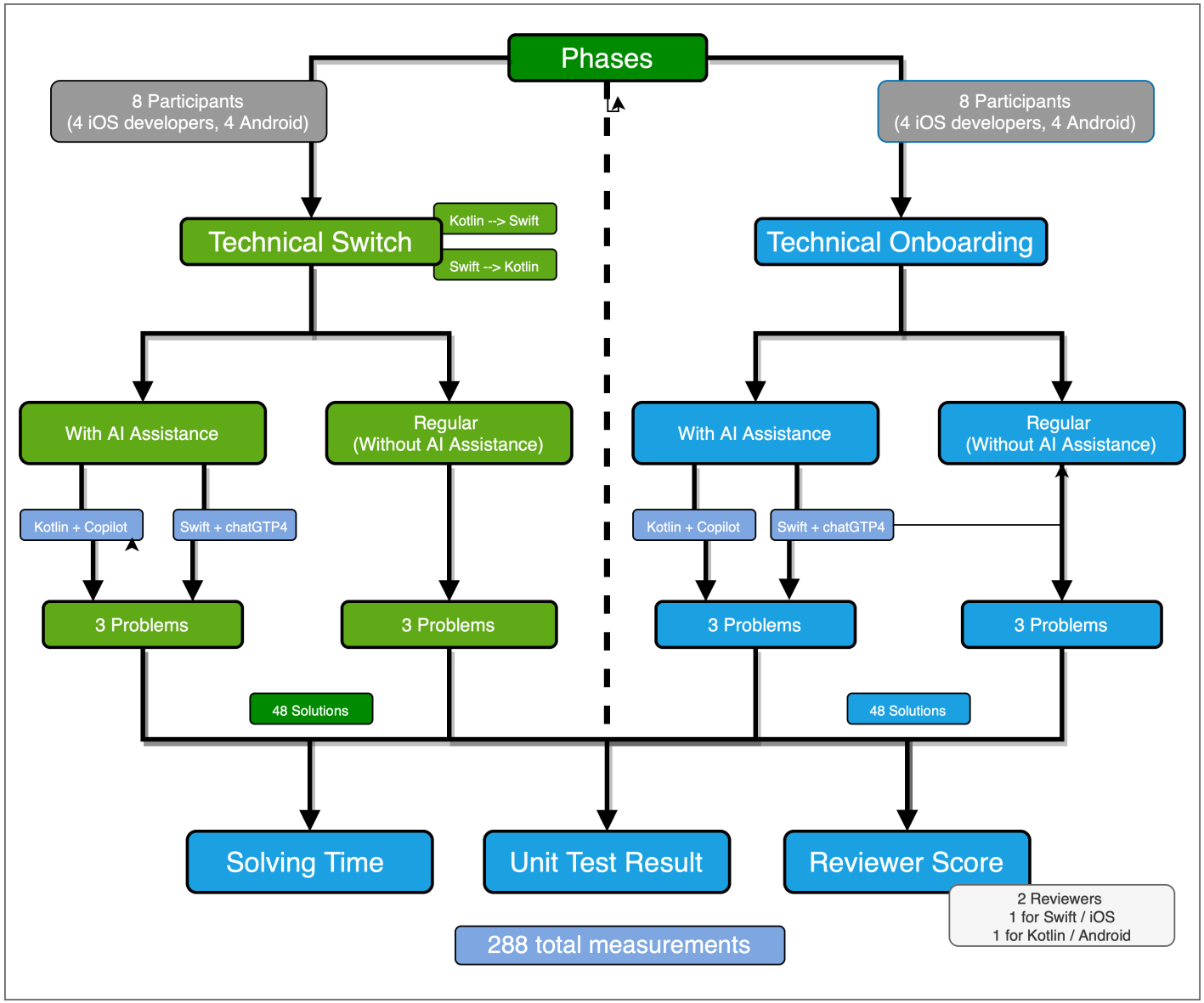
RQ1. How can an AI-based code generator affect the experience when onboarding a new team member or switching technical stacks of an existing colleague?

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

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

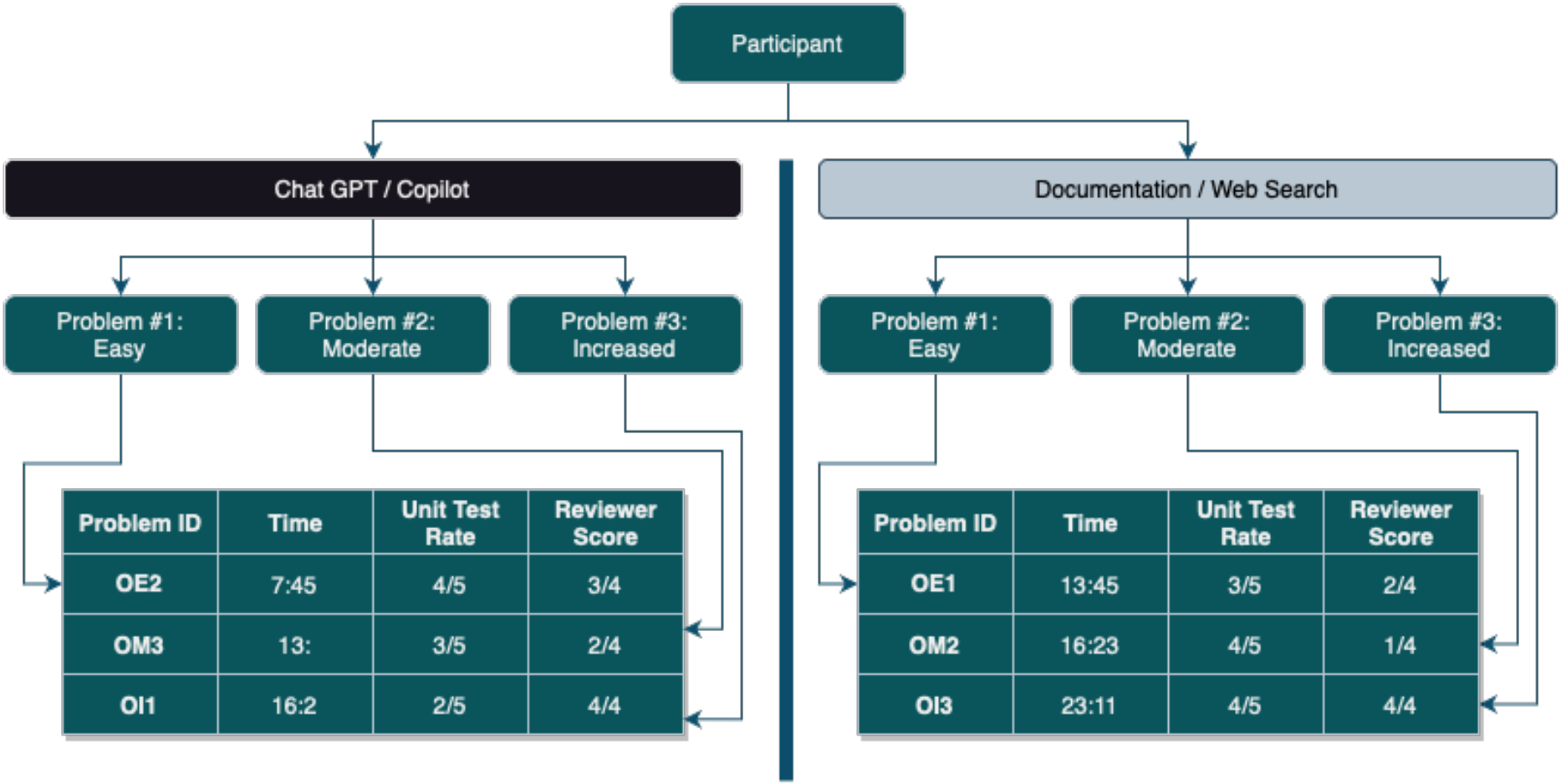
	Technical Onboarding	**Technical Stack Switch**
AI Tool Used for Kotlin	Github Copilot	Github Copilot
AI Tool Used for Swift	ChatGPT	ChatGPT
Total number of distinct problems	9	9
Number of participants	8	8
Number of problems per participant	6	6
Number of problems using AI Tool	3	3
Total Number of solutions generated	48	48
Number of metrics resulted per solution	3	3

Case Study: Using AI-Assisted Code Generation - Overview



Case Study: Using AI-Assisted Code Generation - Results

Quantitative Results



Problem ID	Time	Unit Test Rate	Reviewer Score
OE2	7:45	4/5	3/4
OM3	13:	3/5	2/4
OI1	16:2	2/5	4/4

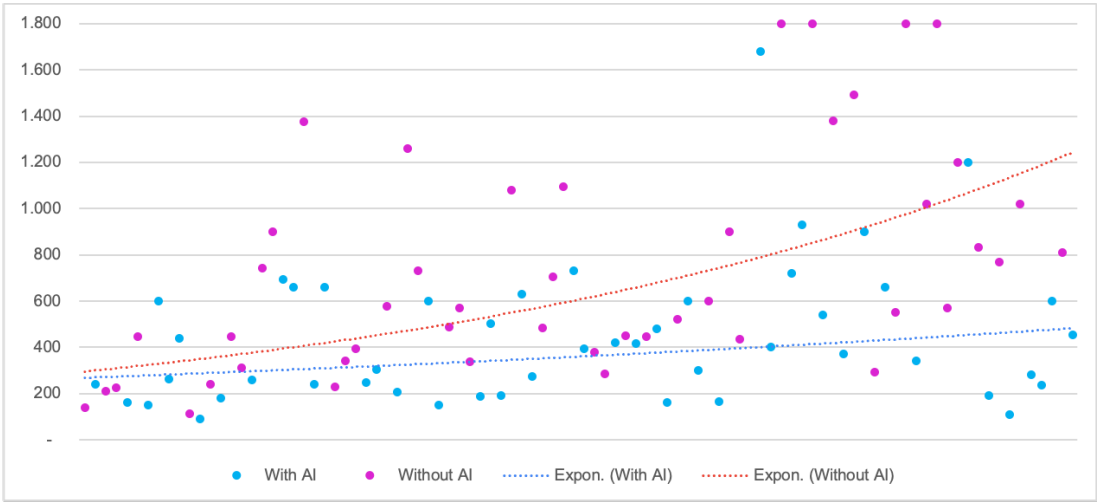
Problem ID	Time	Unit Test Rate	Reviewer Score
OE1	13:45	3/5	2/4
OM2	16:23	4/5	1/4
OI3	23:11	4/5	4/4

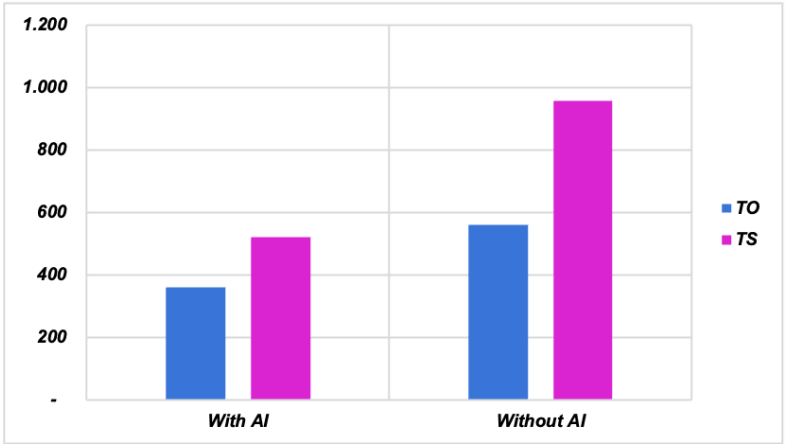
Case Study: Using AI-Assisted Code Generation - Results

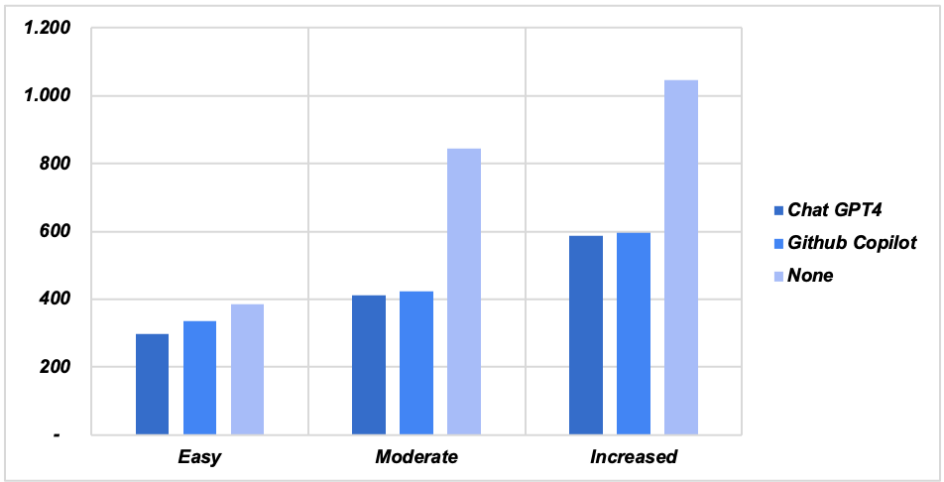
Qualitative Results: Post-Tasks Survey and Reviewer Observation

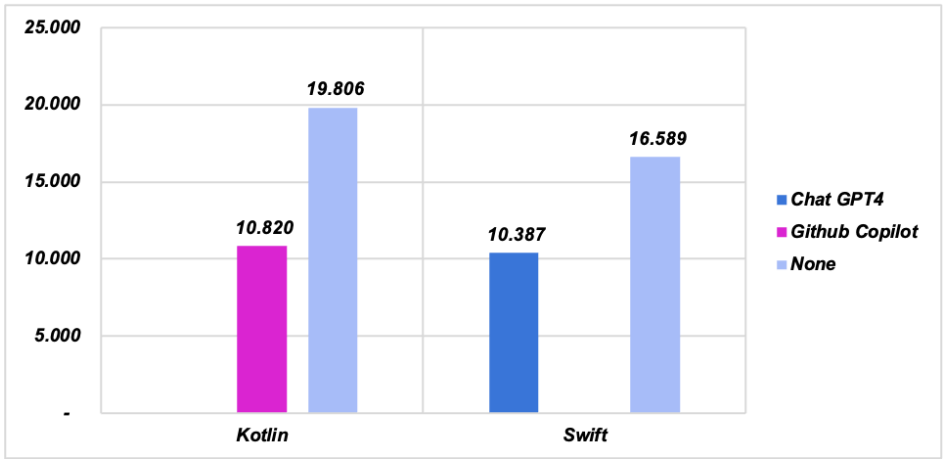
1. Rate the **helpfulness** of the AI-assisted tools (GitHub Copilot/Chat GTP) on a scale from 1 to 5.
2. Rate the level of **understanding** of the AI-assisted code on a scale from 1 to 5.
3. Rate the level of **confidence** you have in the final code on a scale from 1 to 5.
4. Rate the level of expected **future usage** of the AI-assisted code tools on a scale from 1 to 5.

Case Study: Using AI-Assisted Code Generation - Results

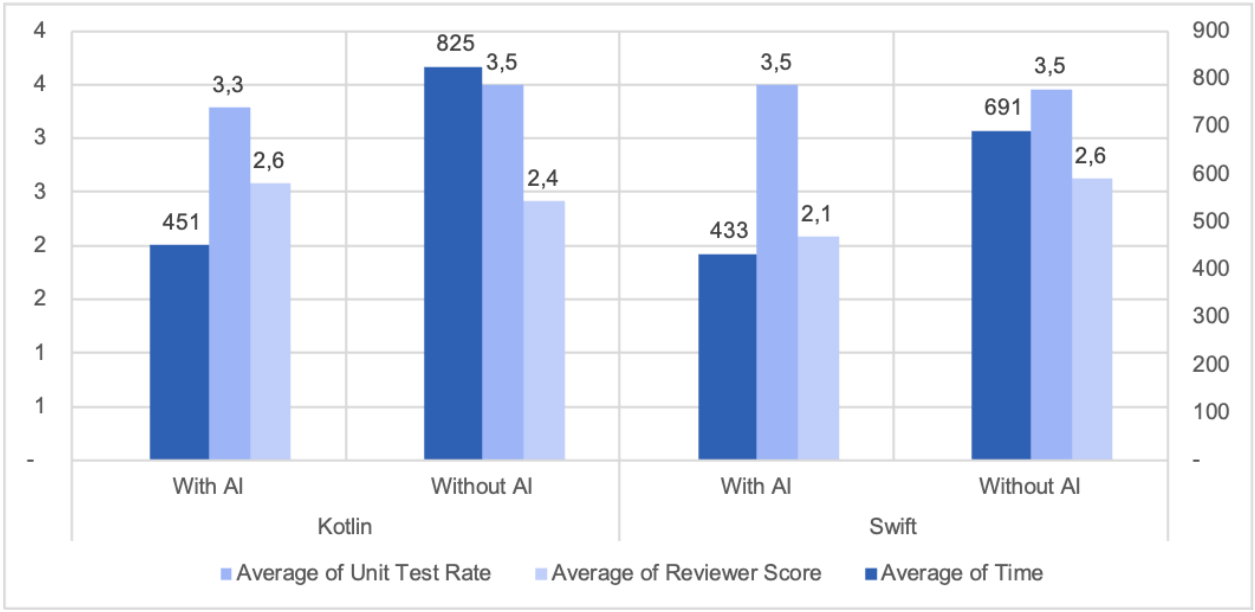


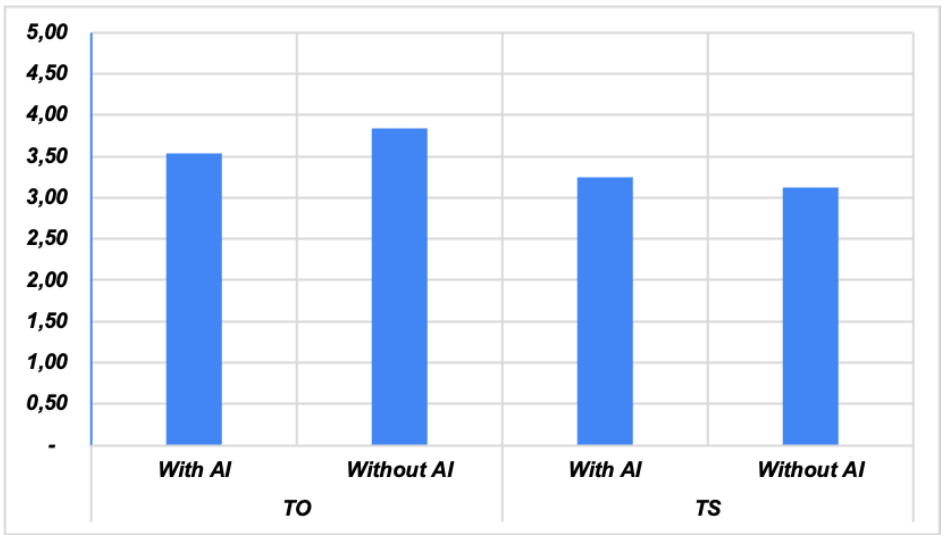


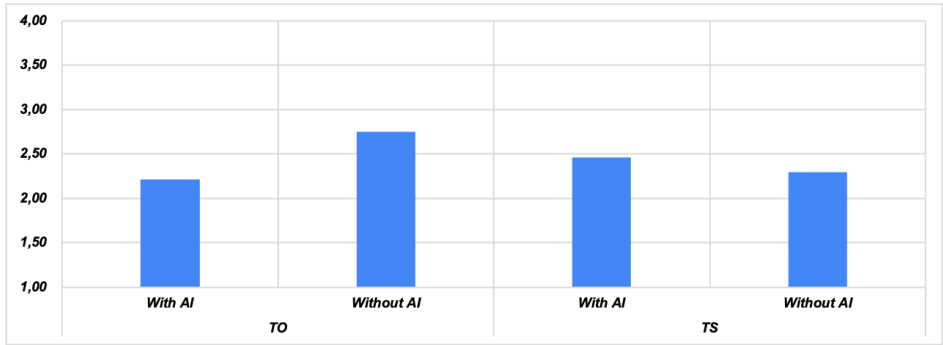




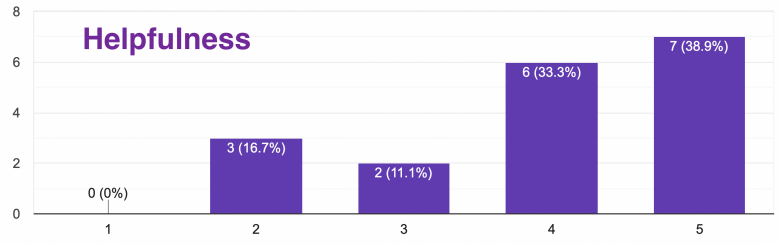
Case Study: Using AI-Assisted Code Generation - Results

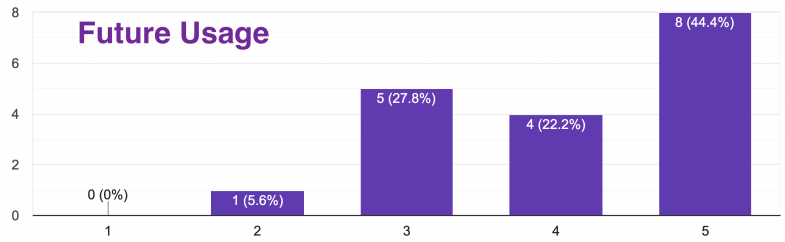


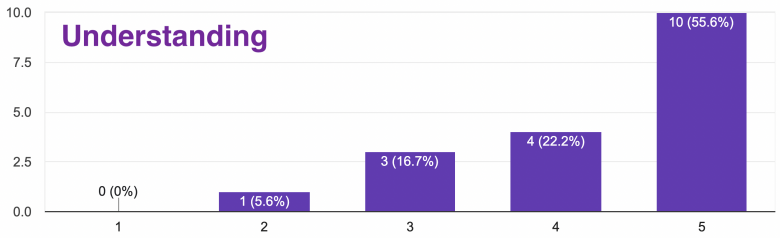


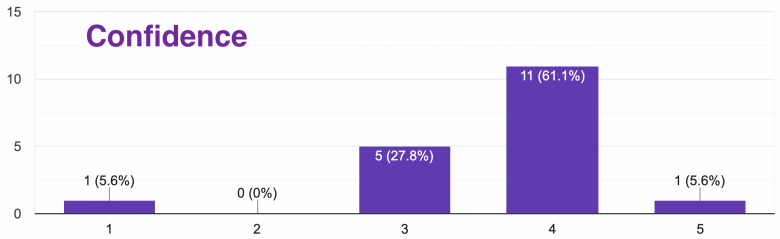


Case Study: Using AI-Assisted Code Generation - Results









1. Adopting the AI Tools Takes Time
2. Quality Prompts Matter
3. Incremental Prompts Improved Correctness and ReviewerScore
4. Platform Association/Conversion Enhances All Metric

Case Study: Discussion

1. AI-Based code generators can **improve** the experience by providing an additional source of information
2. 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
3. Impact of using such tools **can** lower the alignment due to reliance on a tool and less on the team's context and requirement.
4. Effective prompt engineering lead to higher scores.

Conclusions

1. Methodology for dataset creation based on GitHub extraction and Curating can assist in particular code-related requirements for LLMs.
2. The resulting model is successful at obtaining specific platform sample code. Other approaches should be considered.
3. Applying AI-Assisted code generators, at least in non-sensitive areas of a mobile department, can have a positive impact on productivity.
4. Further research and development is planned.