Test Migration Across Mobile Apps Based on Conversational Large Language Model

Serena Liu
Shanghai High School International Division
Shanghai, China
serenalsr0611@gmail.com

Joey Li
Shanghai High School International Division
Shanghai, China
joeyli.shsid@gmail.com

***Abstract*—GUI-based testing is commonly utilized to assess the functionality and usability of mobile apps. Despite the plethora of GUI-based test input generation techniques proposed in the literature, they still face several limitations: (1) lack of context-aware text inputs, (2) absence of test oracles, and (3) being costly and time-consuming. To overcome these shortcomings, we present CHAGRATION, a framework that combines information retrieval and semantic mapping techniques, accompanied by ChatGPT, a conversational large language model (LLM), to migrate test cases and oracles for testing other apps with similar functionalities.**

We first test LLM's ability to map low-level one-to-one event pairs and reached an accuracy of 97%, proving the ability of LLMs for test migration but also showing the necessity of providing context. We then test CHAGRATION on 40 test pairs across ten apps under three different levels of goal summarization. When compared with the state-of-the-art migration tool CRAFTDROID, CHAGRATION can reach a higher accuracy while using significantly less time. We also discovered that the level of summarization will affect the overall accuracy under different types of tasks. Overall, our evaluation of CHAGRATION on real-world commercial Android apps validates its effectiveness and efficiency in transferring both GUI events and oracles.

***Index Terms*—Automation GUI Testing, Test Migration, Large Language Model**

I. INTRODUCTION

As mobile applications have become ubiquitous and play a pivotal role in various aspects of daily life, ensuring their reliability and functionality is of paramount importance. One of the key challenges in guaranteeing the quality of mobile apps is thorough testing. However, manual test case development for mobile apps is often expensive and labour-intensive, requiring significant human effort to define test cases and verify their results.

Due to the challenges of manual test case development and to reduce testing costs, researchers have delved into the realm of automatic GUI test generation. GUI testing involves designing and executing test cases that consist of sequences of events interacting with the graphical user interface (GUI) and assertion oracles that predicate the GUI state. Several approaches have been proposed for automatically generating GUI test cases, with varying degrees of success.

Existing GUI test generators suffer from certain limitations. Many current approaches either generate semantically meaningless tests or rely on implicit oracles that miss failures related to semantic issues. Moreover, as the size of the execution space in GUI applications is vast, achieving comprehensive test coverage is a challenging task.

To address these limitations and explore more efficient testing strategies, research has introduced the concept of test case migration across similar mobile applications. The idea is to leverage similarities between apps to migrate test cases from a source app, which shares similar functionalities, to a target app. By doing so, the test case migration process aims to generate semantically meaningful GUI tests that exercise the functionalities of the target app while adapting relevant Oracle assertions. This approach can potentially overcome the limitations of traditional GUI test generation and provide more effective and cost-efficient testing solutions.

The basis of test case migration lies in the observation that many GUI applications share semantically similar functionalities, even though their GUIs may exhibit significant differences. Apps belonging to the same category, such as banking applications, often share common functionalities and inherently similar GUIs. By mapping semantically similar GUI events and components, automatic approaches can successfully migrate GUI tests across apps, enabling better coverage of the application's functionalities and revealing potential faults. However, currently state-of-the-art tools are still relatively expensive and time-consuming.

The recent advances in Large Language Models (LLMs) provide a new pathway to achieve semantic test migration. The most notable one is ChatGPT [1]. ChatGPT operates based on the transformer architecture, a powerful neural network design that enables it to generate coherent and contextually relevant responses. It utilizes a sequence-to-sequence paradigm, where it processes input text through an encoder and then generates output text using a decoder. The model is pre-trained on vast amounts of text data and can be prompt-engineered for specific tasks, making it adaptable for various applications. Prompt engineer refers to constructing input prompts to the LLMs so that the LLMs can produce more accurate, relevant, and useful output [2]. By carefully prompt-engineering our inputs, we can exploit ChatGPT's potential to perform test case migration across mobile apps [3] [4] [5] [6].

To address the current disadvantages of automated test case migration and enhance the effectiveness and efficiency of LLMs, we propose a cost-effective approach called CHAGRATION. We evaluated it involving 10 Android apps. Our results show that CHAGRATION successfully adapts semantically

Tencent SSV

relevant GUI tests, including oracles, confirming that LLMs-based test case migration generation is a promising and complementary solution for generating GUI tests. Experimental results show that ChaGration successfully migrated 90% of migration pairs within the Level 2 abstraction level, surpassing the state-of-the-art tool CraftDroid by 13.5%. This demonstrates the feasibility of leveraging LLMs for automated test case migration and the effectiveness of ChaGration.

In summary, the contributions in this paper can be described as follows:

• ChaGration, an automated GUI events migration approach using ChatGPT.

• Conducting preliminary studies that validate the effectiveness of LLMs for low-level UI event mapping and the need for context when using LLMs for migration.

• Evaluations that demonstrate the effectiveness of ChaGration in generating migrated test case compared to CraftDroid.

II. Preliminary

In this section, we will introduce basic concepts relating to the formulation of our problem and introduction of ChaGration.

A. Android UI System

UI Hierarchy. Android UIs can be structurally represented in the hierarchical relationship of different UI elements on a screen, including parent, sibling, and child relationships. Individual UI elements contain UI properties such as location, size, and others.

UI Events. An event comprises an actionable UI widget and an associated action. The action may consist of a text input (e.g., user input in the text box). We denote each event as (w, a) as w represents the UI widget and a represents the action type.

UI State. A UI state describes the state the App is currently in. A UI event may change the UI state of the App. For simplicity, we use screen and UI states interchangeably.

UI test case. Every test case exercises a given functionality in an app, such as *sign-in*. A feature-based test consists of a sequence of UI events. A UI test case consists of a sequence of UI events $(w_1, a_1), (w_2, a_2), ...(w_n, a_n)$ and the UI states $(s_1, s_2, ...s_n, s_{n+1})$. s_1 is the UI state before (w_1, a_1). In other words, it is the initial UI state for the App Under Test (AUT). In the paper, we use goal, feature, and intention interchangeably to avoid confusion.

Test Case Migration. When we have a UI test case from a source test, we can use semantic similarity to migrate the UI events to a target app so that similar functionality is exercised in the target test. A target test case will be produced after the migration.

B. Large Language Models and Prompt Engineering

Large Language Models. The recent advances in large language models (LLMs for short) have created paradigm shifts in the natural language processing community. Recent LLMs utilize the Transformer architecture and train on a vast amount of data. It shows great ability in content understanding and generation.

Prompt Engineering. When requesting an output from an LLM, the format of the output may be undesirable and varied. In prompt engineering, we manipulate the original input x into x' that allows the LLMs to better understand the prompt through a function $f_{function}(x) = x'$ that maps x to x'. Through prompt engineering, LLMs can produce output with better effectiveness compared to using x directly. An example of using prompt engineering to produce better results is shown in Fig. 1. By adding a ”TL;DR:” at the end of the prompt, the LLM can summarise the input without more instructions.

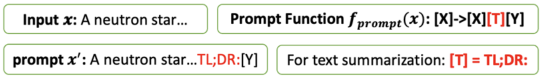
Input x: A neutron star...
Prompt Function $f_{prompt}(x)$: [X]->[X][T][Y]
prompt x': A neutron star...TL;DR:[Y]
For text summarization: [T] = TL;DR:

Fig. 1. An example of prompt engineering

III. Motivating Example.

This paper is organized as follows: Section III introduces a motivating example for our usage of LLMs. Section IV presents the approach for our technique. Sections V and VI present the evaluation results.

The current migration tool mostly adopts a deep-learning approach to link the source event with the targeted event. However, the tools will not consider the semantic meaning of the UI states or the events but solely focus on the semantic similarity. This will lead to the tools trap inside a UI state until there are no more unperformed events left for the UI state. Consider App douzifly in Fig 2. When using CraftDroid, a state-of-the-art migration tool, to create a new to-do list, instead of finishing the migration by clicking the checkmark button in the first picture, CraftDroid decides that the remainder button is more similar to the checkmark button and turns into the remainder page in the second picture. Then it will try to generate an oracle based on the source test and the UI elements on the screen. It will continuously click on every available widget on the screen until it realises there are no desirable widgets and return to the first picture. Eventually, it selects the correct checkmark button. This motivating example highlights the underlying inefficiency in the current migration tools: the failure to identify the semantic meaning of the widgets. This issue causes CraftDroid to fail to realize it had been performing actions on widgets that are nearly the same. On the other hand, LLMs can interpret and compare the current widgets on the screen, and produce an optimal decision.

We observe that most of the current state-of-the-art migration tools-CraftDroid, AdaptDroind, AppTestMigrator, and Screen2Vec-use word embedding techniques (Word2Vec, WMD) to compare the similarity between words. This is often costly because it requires time and hardware to train such neuron networks. Conversational Large Language

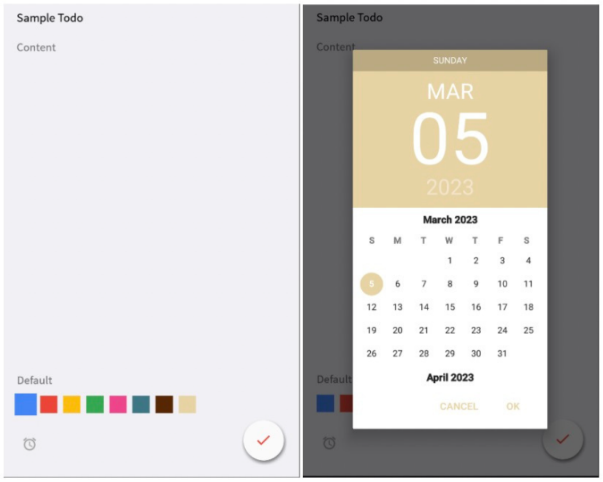

Fig. 2. A motivating example of the inefficiency of current migration tools

Models (LLMs), such as the state-of-the-art ChatGPT, can avoid such a problem as they are already trained. Unlike traditional approaches that involve running models locally, our method capitalizes on the advantages of ChatGPT's cloud-based infrastructure, significantly reducing computational costs and rendering the process more economical. LLMS can identify synonyms, such as ”Login In” and ”Sign Up”, which is often overlooked by existing tools [12]. Understanding synonyms can positively affect the accuracy of comparing similar actions. Additionally, LLMs can view the information of UI elements in a more cohesive way (focus on the whole meaning instead of the meaning of individual words), which makes them potentially better than existing word embedding techniques which can sometimes favour longer sentences with more words [12].

IV. Approach Overview

In this phase, we employ the technique of prompt engineering to make LLM pretends to be an Android UI Testing Expert, helping us to perform Android UI Migration. ChaGrator's design comprises four pivotal modules, each playing a distinct role in the seamless UI test case migration process. Firstly, the **Target Interpretation module** receives and translates information from the source test case software into natural language descriptions, enabling LLM to comprehend the migration target effectively. Subsequently, the **Decision-Making module** utilizes historical performance data and generated summarizations, accompanied by feature-based communications, to prompt LLM to make informed decisions about UI element selection and actions. With the decisions in hand, the **Implementing Responses module** takes charge, executing LLM's previous response on the AUT while generating migration information and verifying the degree of completion. Lastly, the **Context Refinement module** records interactions and dynamically addresses encountered failures, including cycling and infinite scrolling, fostering adaptability and avoiding traps throughout the migration. The main structure of ChaGration is shown in Fig. 3.

A. Target Interpretation Module

Algorithm 1 (lines 6-12) shows how Test Interpretation Module works. When migrating a test case from a Source App, $srcApp$, to a Target App, $tgtApp$, the test case t in $srcApp$ includes several GUI events denoted as $t_{src} = (w_1, a_1), (w_2, a_2), ...(w_n, a_n)$, where w_i represents a GUI widget, and a_i represents the action performed on w_i. Among these GUI events, they have UI Hierarchy $S_{src} = (s_1, s_2, ...s_n, s_n + 1)$. Here, s_n represents the UI Hierarchy before the action of (w_n, a_n), and s_{n+1} represents the UI Hierarchy of the screen after performing (w_n, a_n). To enable LLMs to comprehend the purpose of the source test case more effectively, ChaGration employs a comprehensive approach. It not only distils the source test case but also extracts relevant textual information from the UI hierarchy, thereby avoiding the probability of misinterpreting the intention of the source test case. Subsequently, ChaGration translates this amalgamation into coherent and natural language descriptions, enhancing the likability for LLMs to interpret the information thoroughly.

1) **Distillations:** Providing all contexts to LLM can be distracting due to unnecessary and irrelevant information, such as class, package, and enabled status [13] [14] [15]. After thorough exploration, we have decided to focus solely on widgets that are *clickable*, *long-clickable*, *checkable*, *scrollable*, or contain text when processing the UI Hierarchy. If any of these conditions are met, we will also incorporate processed *resource-id* into the UI Hierarchy description. Additionally, while processing the source test case, we include information such as processed *resource-id*, *parent text*, *sibling text*, *text*, and *actions*. Processing resource-id is essential because it often contains specific information about the current app, which goes beyond the intended textual information of the interacting widget. For instance, in a shopping app named ”fivemiles,” a resource-id like ”com.thirdrock.fivemiles:id/input_layout_email” may contain extraneous elements like ”com.thirdrock.fivemiles:id/” and unnatural underline characters. Consequently, we process it into an ”input layout email”, clearly indicating its purpose for users to enter their email and enabling LLM to understand it more easily. Finally, we processed the original source test case $t_{src} = (w_1, a_1), (w_2, a_2), ...(w_n, a_n)$ into $t'_{src} = (w'_1, a_1), (w'_2, a_2), ...(w'_n, a_n)$, and original UI states $S_{src} = (s_1, s_2, ...s_n, s_{n+1})$ into $S'_{src} = (s'_1, s'_2, ...s'_n, s'_{n+1})$.
2) **Summarization:** When demonstrating the UI Hierarchy, our approach aims to depict the layout and hierarchical structure of the screens accurately. Instead of merely providing a list of indexes within the screen, we utilize indentations between adjacent UI widgets, where each indentation signifies a specific level in the hierarchy.

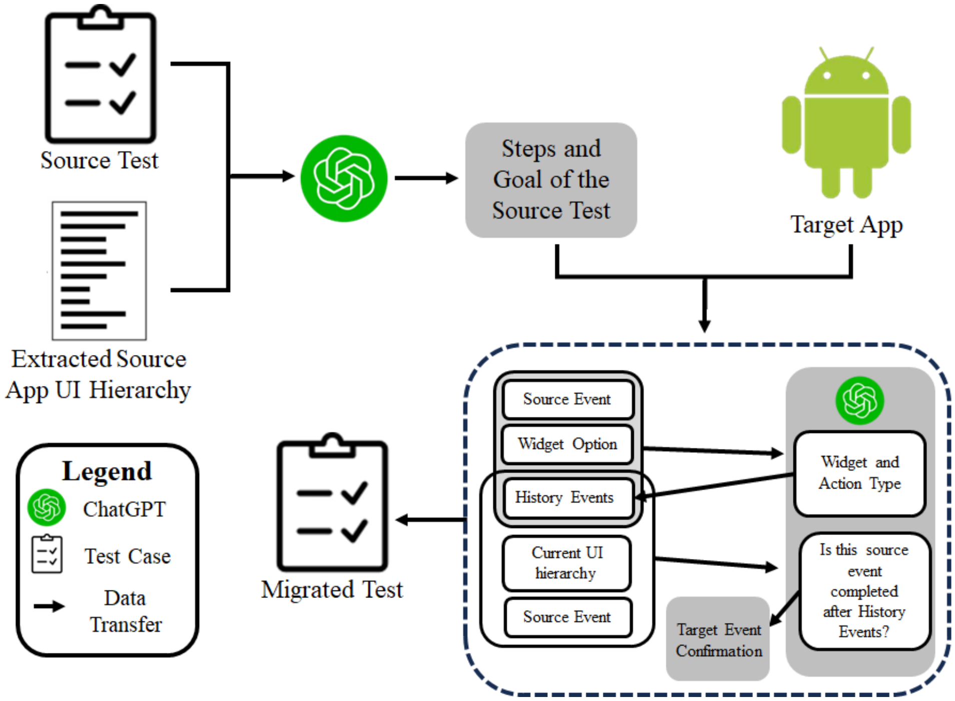

Fig. 3. Design of our tool

This allows us to establish parent-child relationships between elements.

Finally, we describe the source test case using the following template: ”This is the current UI Hierarchy. Indentations indicate the hierarchy level: $s_1^{'}$. Now, we execute this event within the current UI Hierarchy: $(w_1^{'}, a_1)$...... This results in the final UI Hierarchy, indentations represent the hierarchy level: $s_{n+1}^{'}$. ” By interpreting the information, LLM generates a broad summary of the overall goal of this source test case.

B. Decision-making Module

Algorithm 1 (lines 13-36) illustrates the functionality of the Decision-making Module. Instead of simply instructing the LLM to execute the overall goal of the test case, we adopt a more precise approach by requesting it to perform events within the source test case in a chronological sequence. We initialize the index as 1 at the beginning and sequentially increment the index when each GUI event is completed. Additionally, we provide the LLM with the overall goal of the source test case extracted through the target interpretation module. The GUI events for the source test case contain two different categories, GUI actions and Oracle actions, and we withstand them through different approaches.

GUI. For GUI-related source events, we task the LLM with selecting the most suitable element to carry out the current event using a feature-based strategy. Our candidate list comprises all *clickable*, *scrollable*, and *checkable* items, accompanied by essential information such as *child text*, *parent text*, *siblings text*, *content-desc*, and processed *resource-id*. It is worth noting that the inclusion of child text is a novel approach, not required in other contemporary state-of-the-art tools. This additional information becomes particularly valuable when interpreting UI elements that lack other text messages. For example, within Fox News’ settings UI elements, we discovered that the ”about” information could only be acquired through child text, rather than other text messages. After acquiring these candidates, we list the UI element description by using the feature-based strategy, with each one in the form “index*: [description]”. After LLMs selects the element, we provide a decided performance for this index to act on. Our decision includes 1) Click 2) Long-Click 3) Send keys 4) Send keys and enter 5) Send keys and search 6) Clear and send Keys 7) Swipe, incorporating all actions we’ve already encountered within the contemporary test case.

Oracle. For Oracle-related source events, we currently consider five distinctive oracles. These oracles are specifically included in CraftDroid and are as follows: *assertEqual(VALUE, attr(w_i))*, *elementPresense(w_i)*, *elementInvisible(w_i)*, *textPresense(STRING)* , *textInvisible(STRING)*. We first directly imitate the oracle performances done within the source test case. If an error appears, we will revert back to the previous GUI events, as it indicates that the previous action is not desirable enough.

However, the strategy of displaying source test cases one by one in chronological order contains disadvantages apart from advantages. On the positive side, this specific and explicit goal-setting enhances the likelihood of the LLM accurately selecting the appropriate buttons and completing the test case migration with minimal setbacks. The clarity in instructions aids the LLM in understanding the task at hand and reduces

Algorithm 1 Main Algorithm of Our Tool.

```
    Input: Source App UI State $s_{scr} = \{s_1, s_2, s_3, ..., s_n\}$
    Input: Source Test Case $t_{scr} = \{t_1, t_2, t_3, ..., t_n\}$
    Input: Target App tgtApp
    Output: $t_{tgt}$
 1: function CHATGPT_GEN(task, information, requirement)
 2:     prompt ← PROMPT_GEN(task, information, requirement)
 3:     content ← CHATGPT(prompt)
 4:     return content
 5: end function

 6: function GET_GOAL($s_{scr}$, $t_{scr}$)
 7:     task ← "Summarize the test case"
 8:     information ← $s_{scr}$, $t_{scr}$
 9:     req ← "The goal should be in the format F"
10:     Goal ← CHATGPT_GEN(task, information, req)
11:     return Goal
12: end function

13: function MIGRATOR($s_{scr}$, $t_{scr}$)
14:     Goal ← GET_GOAL($s_{scr}$, $t_{scr}$)
15:     index ← 1
16:     history ← ∅
17:     current_src_event ← $t_{scr}$.get(index)
18:     current_UI ← UI_EXTRACTOR
19:     while IS_FINISHED(Goal, history, current_UI) do
20:         Info ← current_UI, history, current_src_event
21:         req ← "Choose one widget"
22:         Widget ← CHATGPT_GEN(Goal, information, req)
23:         Option ← ACTION_EXTRACTOR(Widget)
24:         Action ← FIND_ACTION(Widget, Option)
25:         PERFORM_ACTION(Action, Widget)
26:         history.append(Action)
27:         current_UI ← UI_EXTRACTOR
28:         if CHECKSRCEVENT(current_src_event, history) then
29:             if index<t_scr.size() then
30:                 index ← index + 1
31:                 current_src_event ← $t_{scr}$.get(index)
32:             end if
33:         end if
34:     end while
35:     return TEST_GEN(history)
36: end function

37: function IS_FINISHED(Goal, history, tgtUI)
38:     task ← "Is Goal reached in the target app"
39:     information ← Goal, history, tgtUI
40:     req ← "Answer Yes or NO"
41:     return CHATGPT_GEN(task, information, req)
42: end function
```

the chances of misinterpretation. However, on the other hand, challenges may arise when encountering None-to-None migrations, where the events from the source test case do not exist within the target app. In such scenarios, the LLM might face difficulties and lose track of the intended sequence. A notable example involves software platforms with varying welcome sessions. Some software may include feature explanations, while others may prompt users to authorize personal information. In these cases, pretreatment becomes imperative to consider.

We implement the following pretreatment: when the migration process moves to migrate $(w_i^{'}, a_i)$, we would first ask LLM if "this particular GUI event is useful to reference or commonly applied in order to finish the goal in all other related apps." Specifically, we asked LLM to ignore actions that seemly related to "skip", "close", "allow/not allow" shown in source test cases. LLM could directly ignore welcoming sessions' GUI events and directly move on towards related GUI events.

C. Implementing Responses

After generating actions, we automatically generate the corresponding migration test case in JSON format as well as performing this particular action in AUT. At the same time, we would validate whether the action chosen by LLM above successfully finished the event finished by the source test case, as shown in Algorithm 1 (lines 37-42). If so, we would move on towards the next GUI event to reiterate the step done above until all GUI events are finished. Otherwise, we would continue to finish the current GUI event until we finished.

D. Context Refinement module

In the context of LLMs, which are probabilistic models, and considering that they may not completely understand the implementations of the AUT, erroneous selections of UI elements and failed actions can occur based on the current AUT context. To address this issue, we've implemented a historical store to track elements that are prone to being misunderstood. When a specific UI element is triggered multiple times and appears to be stuck in a repetitive loop, we take preemptive action and block access to that element. Furthermore, to avoid infinite swiping scenarios, we intervene in cases where swiping actions are involved, ensuring that swiping is limited to situations where the page structure is finite.

V. EXPERIMENT

In this paper, we study the effectiveness and limitations of the semantic mapping capability of LLMs, specifically ChatGPT, for test migration. We conducted a set of experiments, aiming to answer the following questions:

- **RQ1.** What is the effectiveness of LLMs in mapping low-level UI events?
- **RQ2.** How can different summarization levels of the intention affect the effectiveness of test case migration?
- **RQ3.** Is CHATGRATION more effective in migration compared to previous work in terms of efficiency and accuracy?

A. Experiment Setup

Subject. The original test cases earmarked for migration were carefully selected from a variety of origins, including ATM, CRAFTDROID, and FRUITER [7] [11] [12]. However, due to the adoption of a test-reuse technique, these tests were captured in disparate formats, making conventional analysis impractical. For example, while ATM's tests were expressed as Espresso [39] tests in Java, CraftDroid's source tests were coded in Python using Appium, and its transferred tests were represented in JSON without the original Python code. To facilitate automated evaluation, standardization of these heterogeneous tests became essential. We extract JSON files from CRAFTDROID and FRUITER and convert them to a standardized format following the structure of CRADTDROID.

TABLE I
Summunary of Apps

Tool	Category	App Name	Tool	Category	App Name
ATM	Expense Tracker	EasyBudget [16]	CraftDroid	To-Do List	Minimal [29]
		Expenses [17]			Clear List [30]
		Daily Budget [18]			Todo List [31]
		Open Money [19]			Simply Do [32]
	Note Taking	Swiftnotes [20]			Shop. List
		Writely Pro [21]		Tip Calculator	TipCalc.Plus
		Pocket Note [22]			FreeTipCalc
	Shopping List	Shop.List1 [23]			TipCalculator
		Shop.List2 [24]			TipCalc [35]
		Shop.List3 [25]			Simple Tip [36]
		OI Shop.List	FrUITeR	Shopping	Five Miles
CraftDroid	Browser	Privacy			Home
		FOSS [27]			geek
		FirefoxFocus [28]		News	abc
	Email	k9			fox
		Mail.ru [34]			usatoday

We check the validity of those test cases recorded in JSON manually and translate them into the Appium script. At last, we extract the UI hierarchy of all the UI states that each test case goes through in an XML form. We only store the actionable elements in the UI hierarchy, including its resource-id, text, sibling information, parent information, and other pertinent attributes. Through these procedures, we successfully generated over 70 test cases by utilizing more than 30 Android apps, categorizing them into 9 distinct categories, as illustrated in Table I.

Test Platform. We implemented ChaGration with Python and for test cases generated using Appium. Existing test cases for the subject apps are written using Appium's Python client and the processed test cases are stored in JSON format. For our experiments, we used an official Nexus 5x Emulator running Android 13.0 (API 33). Each emulator is dedicated with 4 CPU cores and 2 GB of RAM. We used Appium 2.0, ADB, and Android Studio as our platform [40], [41], [42]. The test is run on a Windows computer with 32GB RAM, an i5-12600kf CPU, and one Nvidia GeForce RTX-4070 Graphic Card to ensure the responsiveness of emulators.

Conversational LLM Selection. Although the LLMs can be instantiated with any LLMs, we choose ChatGPT, the most advanced and available conversational LLMs for frequent API requests. While GPT-4 is more effective and outperforms ChatGPT, its response time is relatively slow and unsuitable for large-scale test migration. We use ChatGPT's python API wapper `client.chat.completions.create` with model `gpt-3.5-turbo`, which is faster and cheaper than `text-davinci-003`. The temperature is set to 0.2 such that the experiment is more replicable.

B. Effective of LLMs in Mapping Low-Level UI Events

From our observation of the test cases, there are four major types of event mapping: one-to-one, many-to-one, many-to-one, and none-to-none. By one-to-one mapping, we mean when mapping a source UI event, there is one and only one target UI event that corresponds to the source UI event being mapped from. One such example is the mapping of "press Sign-in" in t_{scr} to "press log-in" in t_{tgt} when migrating a UI event in a pair of test cases that test the login function of an app. On the other hand, high-level event mapping involves one-to-many and many-to-one mapping such that multiple (more or equal to two) source (or target) UI events correspond to one target (or source) UI event. Such high-level mapping is still an ongoing challenge for the future [12]. None-to-none is the most intricate scenario, such as advertisements that are unique to one app.

Since performance on low-level mapping is an important indicator of the capability of migration tools, our experiment on low-level event mapping can allow us to test the fundamental capability of LLMs in test-case migration, including its limitation in certain one-to-one scenarios. Together, we identified and extracted event pairs from all migration pairs that represented one-to-one migrations, resulting in nearly 500 results. Our algorithm is as follows: for example, when migrating a23 to a24, the event pairs are $(s3, t1), (s4, t2), (s5, t3)$, with the first element in each tuple representing an event associated with a23, while the second element represents an event associated to a24. Subsequently, we tasked LLMs with matching the source event with the UI Hierarchy of the target event, simulating the process of one-to-one migration. This was done lacking any contextual information, meaning LLMs are unaware of the overall goal of the test case or any prior actions experienced.

C. The Effect of Different Levels of Summarization on CHATGRATION

When summarizing the overall intention of the t_{scr}, we can summarize the intention in different levels of summarization, or the details of t_{scr} given in the summarizing. If given little detail in the intention, LLMs might fail to migrate the test case because of a lack of information. If given too many details, the information in the intention can be too t_{src}-oriented so that the LLMs can be too focused on replicating the events in t_{src} instead of finding the similarity between app_{scr} and app_{tgt}. We defined three levels of summarization when asking ChatGPT to summarize the intention of t_{src}:

1. Only the overall intention of t_{src}, or the feature that is tested in t_{src}
2. The overall intention and summary of UI events in t_{src}
3. The intention and function of every single UI event in t_{src}

For example, when we execute test case on Minimal, the first abstraction level will generate a simple sentence that describes the entire test case in one line: 'The goal of this test case is to add a new ToDo item with the title "Sample Todo" and verify its successful creation and display on the main screen.'

The second abstraction level will summarize the major steps involved in achieving this goal. For instance, in the to-do list, the test case is divided into three main steps: clicking, entering text, and creating. This would be articulated as 'User clicks on the "Add Todo" button', 'User enters the title "Sample Todo"', and 'User clicks on the "Save" button.'

The third abstraction level provides more detailed explanations of each step, such as 'Click on the "Add ToDo Item" floating action button', 'Enter text "Sample Todo" in the "Title" field', 'Click on the "Make ToDo" floating action button', 'Verify the presence of the newly created ToDo item with the text "Sample Todo" .'

In generating the overall goal, we added the following sentence at the end: "If the test case contains sending keys, please also provide the text value when summarizing." This change was made because without this sentence, LLMs would randomly input values within the text field, resulting in an undesired outcome. For example, without this sentence, the goal generated for experiment b51 of CRAFTDROID was, "the goal is to verify the accurate calculation of the tip and total amount based on user-entered values for bill amount and tip percentage." Because this goal did not explicitly specify the numbers to be input, LLMs would randomly input numbers in the bill and tip areas, making it impossible for us to verify whether the total amount to be paid was calculated correctly. After adding this sentence, LLMs would summarize the goal as follows: "Entering a bill amount of 56.6 and a tip percentage of 15, and verifying that the total amount is correctly calculated as 65.09." With this summarization, LLMs can correctly input the numbers and perform oracle checking.

Meanwhile, during the processes for the first and second abstraction levels, we did not perform oracle detection and inclusion. This is because, according to our algorithm, automatic migration of oracles occurs if the source event we are migrating is detected as belonging to the oracle. Therefore, this only applies to one-to-one or one-to-many migrations of the entire test case, which is what the third abstraction level is about. However, in cases where only the goal is present or when test cases are summarized into the main modules only, instead of one-to-one event summarization, this algorithm is not applicable.

We achieve differences in abstraction levels by modifying GET_GOAL in **Algorithm 1**. Each level of distillation is tested on CHAGRATION to test the impact of the difference in distillation of the summarizing of the intention on the effectiveness.

We selected a total of 40 test migration pairs from two categories of Apps. The specific information is shown in Table II.

D. Evaluation Metric

We evaluate the effectiveness of CHAGRATION through the precision of the migrated test case. Suppose a ground truth test case t contains n GUI events $(e_1, ..., e_n)$ and the generated migrated test case consists of n' UI events $(e_1^{'}, ..., e_n^{'})$. Then the precision=$\frac{m}{n}$, where $m = max_{i \in [0, min(n',n)]} : e + i^i \neq e_i \wedge \forall j < i, e_j^{'} = e_j$

VI. RESULT

A. RQ1

Within the nearly 500 one-to-one event migration pairs we extracted, we randomly sampled 178 pairs to assess the precision of their matching process. Of these 178 samples, 172 successfully underwent migration with a precision rate of 97%. The specific correct and incorrect examples are presented in Table 2, along with the types of software and the reasons for errors. This high accuracy of matching demonstrates the significant advantage of LLM in one-to-one matching.

In examining the six instances that failed to migrate successfully, we found that they primarily stemmed from weak semantic relationships between the actions of both applications involved. For instance, the action 'Open Menu' was required to access the profile in Geek, whereas this was achieved by clicking 'Me' within 5 miles. The semantic disparity led the LLMs to incorrectly match 'Open Menu' with 'Home' instead of 'Me.' In such cases, actions that might require multiple attempts by ordinary individuals could also pose challenges for LLM. Therefore, LLM requires contextual support and multiple attempts to address this issue. For example, when we removed the "home" option and prompted a reselection, the model correctly chose "me."

B. RQ2

The result of the precision of the three levels is presented in Fig. 4.

TABLE II
THE BENCHMARK TEST CASES FOR THE TWO CATEGORIES WE SELECTED

Source	Category	App	Test Step	Avg # Total Events	Avg # Oracle Events	# Migration Pairs
CRAFTDROID	To Do List	a21:Minimal a22:Clear List a23:To-Do List a24:Simply DO s25:Shopping List	1. Click the add task button 2. Fill the task title as ”Sample to do” 3. Click the add/confirm button 4. The task should appear	4	1	5*4*1=20
	Tip Calculator	a51:Tip Calculator a52:Tip Calc a53:Simple Tip Calculator a54:Tip Calculator Plus a55:Free Tip Calculator	1. Input 56.06 as the total bill 2. Input 15% as the Tip rate 3. Calculated the total bill with tip which is 65.09	3.8	1	5*4*1=20

TABLE III
ONE-TO-ONE MAPPING FAILED RESULT

Category	Problem
Browser	No Problem
To-Do List	No Problem
Tip Calculator	No Problem
News	Unable to link ”me” with ”open menu” Linked ”open menu” with ”home” instead of ”me”
Shop	Unable to link ImageButton with Back

Level 1. The Level 1 approach offers the advantage of adaptability in scenarios characterized by relatively incongruent software processes. For instance, in the case of a24, it doesn’t necessitate clicking ”add” to input text; instead, it permits direct data entry. In contrast, for a23, it mandates the initial execution of a ’skip’ command to bypass the instructional interface, followed by the ’add’ operation and subsequent data entry, resulting in two additional steps compared to a24. However, when utilizing Level 1, it defaults to the task as ”The goal of this test case is to add a new ToDo item with the title ’Sample Todo’ and verify its successful creation and display on the main screen,” thereby overlooking the nuanced distinctions among diverse software.

Nevertheless, the drawback of Level 1 is that LLMs may erroneously assume that the entire migration has already been completed, prematurely concluding the entire migration process. For example, during the migration spanning from a21-a23, a23-a22, a23-a25, and a25-a21, LLMs conclude the task completion upon the simple input of ’Sample Todo,’ disregarding the subsequent imperative step of initiating the creation process. Furthermore, during the migration of test cases characterized by a profusion of events, LLMs are susceptible to losing their orientation, although this is not evident in the a2 and a5 instances due to their relatively low average event count.

Level 2. Compared to Level 0, the Level 1 approach is more modular, breaking down a single-sentence goal into several smaller goals. Under this circumstance, ChaGration has a clearer understanding of the test cases that need to be completed. However, the problem arises if the first subgoal is executed incorrectly and is mistakenly judged by LLMs as the correct direction, it will be challenging to get back on track thereafter.

Level 3. In this level of summarization, we require ChatGPT to provide more detailed information on the source test case. Level 3 performs the worst out of the three levels of summarization. Since in this level, the importance of replicating the events in the source test case is emphasized more than the importance of the goal, CHAGRATION may pursue complete individual events instead of the overall goal. For example in a51, the total bill is automatically calculated and does not require clicking the button ”Calculate” like in other Tip Calculator Apps. Thus when migrating the source test case to a51, the precision rate is relatively low as CHAGRATION insist that this step hasn’t been finished.

Likewise, level 3 summarization is also oriented to find the exact UI elements to act on. In some Tip Calculator apps, 56.6 should be entered into the text box ”Total Bill”, and the bill with the tip will appear as ”Bill Amount”. In other apps, the situation is exactly the opposite. In the previous two levels, CHAGRATION was able to identify the difference after several tries, but in this level, CHAGRATION continued to enter 56.6 in the bill with the tip text bar. The over-details of level 3 led to its poor performance.

From the result, it can be shown that the level of summarization must be set at a reasonable limit. More simpler test cases with shorter lengths and less complex apps, a more detailed summarization may be more favorable. But for apps with many functionalities and UI states, a less detailed summarization can bring CHAGRATION closer to the desired goal.

C. RQ3

1) Accuracy: Compared to the state-of-the-art migration tool CRAFTDROID, Fig 4 demonstrates that CHAGRATION can migrate text cases at a much higher accuracy, except for Level 3. In Level 2 To-Do List app, CHAGRATION can reach a precision of 923 %, which is more effective than CRAFTDROID’s precision rate of 74 %. However, both precision of level 3 migration are lower than that of CRAFTDROID.

2) Efficiency: We set the upper limit of each migration to 20 events since the test case we use is relatively short. Out of the 120 migrations, only 3 experiments reached this upper limit, which won’t significantly affect the overall efficiency of CHAGRATION. The efficiency of CHAGRATION is extremely

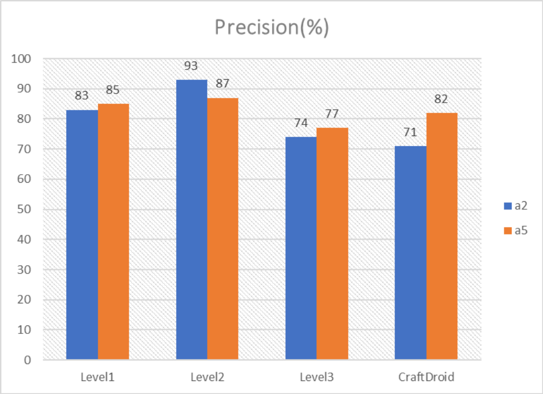

Fig. 4. Comparison of Precision Between CraftDroid and ChaGration

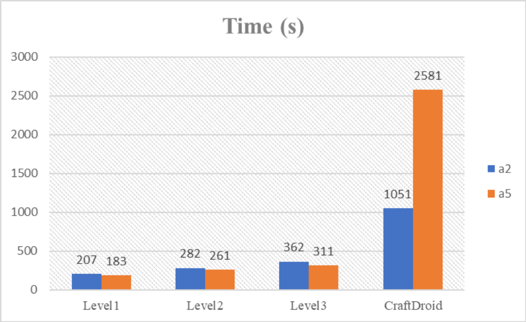

Fig. 5. Comparison of Runtime Between CraftDroid and ChaGration

faster and advantageous than that of CraftDroid. ChaGration spend on average somewhat between one-third to one-eighth of the time used by CraftDroid. Most of the time is spent on waiting for the API request from ChatGPT. A test transfer ranges from 3 minutes to 10 minutes. Such efficiency means that developers don't require multiple emulators to speed up the migration process. Setting up multiple emulators is costly for the hardware requirements of the testing platform. The relative efficiency and easiness of ChaGration makes test migration more available for developers with normal computers.

VII. Threats to Validity

As we were working on this paper, OpenAI released GPT-4V which supported picture input [43]. We apply our current framework to this new LLM model. We do so by removing the textual UI information and replacing it with the screenshot of the current UI. Every actionable UI element is marked with a number like our previous framework. The effectiveness did improve but with the burden of an increasing cost. Each picture input requires a certain resolution and costs 0.02 dollars. Together, every migration will cost about 0.5 dollars, which is too costly for developers. Additionally, GPT-4V is still in preview and is unstable and slow. Thus, we continue to use GPT-3.5 as our main LLM. However, we believe that GPT-4V can be exploited in a way to bring better performance in test case migration. Internally, since ChatGPT is a probabilistic model, we run all migration tests three times to reduce the randomness of the result.

VIII. Related Work

Test migration based on semantic similarity had been a hot research topic. Lin, Jabbarvand, and Malek had developed CraftDroid to migrate test cases. It does so by semantically mapping the actionable GUI widget and searching for the widgets that are most similar in the source and the target app. It leverages word embedding produced by *Word2Vec*, a Natural Language Processing (NLP) neuron network, to compare the similarity of two-word lists [7]. Similarly, Behrang and Orso developed AppTestMigrator (ATM) which shared many commonalities with CraftDroid such as using *Word2Vec* and mapping Window transition Graphs. It improves on the authors' previous work GUITestMigrator [8]. Besides exploiting the word embedding technique, Li et. al used Screen2Vec which is inspired by *Word2Vec*. A self-supervised technique, Screen2Vec aims to use the structure of GUIs, and GUI interaction traces and incorporates screen- and app-specific metadata [9]. Mariani et. al developed AdaptDroid, a technique that uses an evolutionary algorithm to match the GUI test case across apps. It extracts semantic information from a GUI state and uses Word Mover's Distance (WMD) to match test cases between source and target apps [10].

Framework for Evaluation The research into evaluating UI test migration tools is limited, but some researchers have proposed several frameworks and uncovered some key findings regarding semantic test reuse. Zhao et. al presented FrUITeR and applied a uniform benchmark to evaluate the performance of existing migration tools on precision, fidelity, and utility [11]. Mariani et. al conducted an empirical study on how would the corpora of documents, word embedding technique, event descriptors that are extracted, and semantic matching algorithm affected test case migration.

Large Language Models and prompt engineering Large Language Model is a type of Artificial Intelligence that has gained significant attention in recent years. Trained on a vast amount of data, LLMS are a category of foundation models that are capable of understanding and generating natural language, such as translation, summarization, and understanding. Some of the most famous examples include OpenAI's ChatGPT (175B) and GPT-4 (170T) [1] [43], and Google's Gemini (1.6T) [44]. Those multimodal models can also accept both textual image inputs. These new developments in LLMs are used for code generation, bug detection, and parameterization. ChaGration is the first migration tool that can transfer UI events and oracles with LLMs.

IX. Conclusion

Current automated UI testing methods struggle to produce semantically relevant test inputs and related test oracles for the tested application, primarily because UI semantics are challenging and costly to extract using current deep learning approaches. However, the advent of conversational large language models presents an effective and economical means to grasp the UI semantics of both the source and target applications, facilitating the migration of UI test cases autonomously. This paper introduces ChaGration, a system developed for automating UI test case migration using conversational large language models. We implemented ChaGration, utilizing the advanced conversational large language model ChatGPT and utilized the dataset of CraftDroid, a framework that employs information retrieval and both static and dynamic analysis methods for test case migration. The evaluation of ChaGration underscores its practicality in migrating UI test cases using conversational large language models and confirms its efficacy.

Acknowledgement

This work was supported by Tencent Sustainable Social Value (SSV) Organization's Aspiring Explorers in Science Program and was instructed by Prof. Tao Xie and Dr. Ran from Peking University. We would like to thank their generous support in this work. We would like to thank SUSTech for the study environment. We would like to thank Andrew Qu for providing the device and platform for us to perform our tests. We would like to thank Han Su for providing emotional support. We would also like to thank the various help from our peers in the program, which helped us to improve the work.

References

[1] openai. (2023) Introducing to ChatGPT. [Online]. Available: https://openai.com/blog/chatgpt

[2] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," arXiv preprint arXiv:2107.13586, 2021.

[3] F. Petroni, T. Rockta schel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?" *arXiv preprint arXiv:1909.01066*, 2019.

[4] T. Schick and H. Schu tze, "Exploiting cloze questions for few shot text classification and natural language inference," *arXiv preprint arXiv:2001.07676*, 2020.

[5] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," *arXiv preprint arXiv:2104.08691*, 2021.

[6] L. Cui, Y. Wu, J. Liu, S. Yang, and Y. Zhang, "Template-based named entity recognition using bart," *arXiv preprint arXiv:2106.01760*, 2021.

[7] Jun-Wei Lin, Reyhaneh Jabbarvand, and Sam Malek. 2019. Test Transfer Across Mobile Apps Through Semantic Mapping. In Proceedings of the International Conference on Automated Software Engineering (*ASE'34*). IEEE Computer Society, 42-53.

[8] Farnaz Behrang and Alessandro Orso. 2020. AppTestMigrator: a tool for automated test migration for Android apps. In Proceedings of the International Conference on Software Engineering (*ICSE DEMO '20*). ACM, 17-20.

[9] Toby Jia-Jun Li, Lindsay Popowski, Tom M. Mitchell, and Brad A. Myers. 2021. "Screen2Vec: Semantic Embedding of GUI Screens and GUI Components," arXiv.org. https://arxiv.org/abs/2101.11103

[10] Leonardo Mariani, Mauro Pezzè, Valerio Terragni, and Daniele Zuddas. 2021. An Evolutionary Approach to Adapt Tests Across Mobile Apps. IEEE Xplore, 70–79.

[11] Yixue Zhao, Justin Chen, Adriana Sejfia, Marcelo Schmitt Laser, Jie Zhang, Fed-erica Sarro, Mark Harman, and Nenad Medvidovic. 2020. FrUITeR: a framework for evaluating UI test reuse. In Proceedings of the Joint Meeting on Foundations of Software Engineering (*ESEC/FSE 20*). 1190-1201.

[12] Leonardo Mariani, Ali Mohebbi, Mauro Pezzè, and Valerio Terragni. 2021. Semantic matching of GUI events for test reuse: are we there yet? (*ISSTA 2021*) DOI:https://doi.org/10.1145/3460319.3464827

[13] *Android Components*, Google, 2023. https://developer.android.com/guide/components/fundamentals

[14] Google. (2021) Android View. [Online]. Available: https://developer.android.com/reference/android/view/View

[15] Google. (2021) Android ViewGroup. [Online]. Available: https://developer.android.com/reference/android/view/ViewGroup

[16] Benoit Letondor. 2021. EasyBudget. https://play.google.com/store/apps/details?id=com.benoitletondor.easybudgetapp

[17] Luan Kevin Ferreira. 2021. Expenses.https://play.google.com/store/apps/details? id=luankevinferreira.expenses. Last access: Jan 2021

[18] Kvannli. 2021. Daily Budget. https://play.google.com/store/apps/details?id=com.kvannli.simonkvannli.dailybudget. Last access: Jan 2021

[19] xorum. 2021. Open Money Tracker. https://play.google.com/store/apps/details? id=com.blogspot.e_kanivets.moneytracker. Last access: Jan 2021.

[20] Adrian Chifor. 2021. Swiftnotes. https://play.google.com/store/apps/details?id= com.moonpi.swiftnotes. Last access: Jan 2021.

[21] plafu. 2021. Writeily Pro. https://f-droid.org/en/packages/me.writeily. Last access: Jan 2021.

[22] roxrook. 2021. Pocket Note. https://github.com/roxrook/pocket-note-android. Last access: Jan 2021

[23] Andrzej Grzyb. 2021. Shopping List. https://play.google.com/store/apps/details? id=pl.com.andrzejgrzyb.shoppinglist. Last access: Jan 2021.

[24] Vansuita. 2021. Shopping List. https://play.google.com/store/apps/details?id=br. com.activity. Last access: Jan 2021.

[25] SECUSO Research Group. 2021. Shopping List (Privacy Friendly). https://play.google.com/store/apps/details?id=privacyfriendlyshoppinglist.secuso.org.privacyfriendlyshoppinglist. Last access: Jan 2021.

[26] OpenIntents. 2021. OI Shopping list. https://play.google.com/store/apps/details? id=org.openintents.shopping. Last access: Jan 2021.

[27] Gaukler Faun. 2021. FOSS Browser. https://f-droid.org/en/ packages/de.baumann. browser/. Last access: Jan 2021.

[28] Mozilla. 2021. Firefox Focus. https://play.google.com/store/apps/details?id=org.mozilla.focus. Last access: Jan 2021.

[29] Ruben Roy. 2021. Minimal. https://f-droid.org/en/packages/com .rubenroy.minimaltodo/. Last access: Jan 2021.

[30] douzifly. 2021. Clear List. https://f-droid.org/en/packages/douzifly.list/. Last access: Jan 2021

[31] SECUSO Research Group. 2021. Todo List. https://f-droid.org/en/packages/ douzifly.list/. Last access: Jan 2021.

[32] keith kildare. 2021. Simply Do. https://f-droid.org/en/packages/kdk.android. simplydo/. Last access: Jan 2021.

[33] keith kildare. 2021. Shopping List. https://f-droid.org/en/packages/com.woefe.shoppinglist/. Last access: Jan 2021.

[34] Mail.Ru Group. 2021. Mail.ru. https://play.google.com/store/apps/details?id=ru.mail.mailapp. Last access: Jan 2021.

[35] Apps By Vir. 2021. Tip Calc. https://play.google.com/store/apps/details?id=com.appsbyvir.tipcalculator. Last access: Jan 2021.

[36] TLe Apps. 2021. Simple Tip Calculator. https://play.google.com/store/apps/ details?id=com.tleapps.simpletip calculator. Last access: Jan 2021.

[37] ZaidiSoft. 2021. Tip Calculator Plus. https://play.google.com/store/apps/details? id=com.zaidisoft.teninone. Last access: Jan 2021.

[38] JPStudiosonline. 2021. Free Tip Calculator. https://play.google.com/store/apps/ details?id=com.jpstudiosonline.tipcalculator. Last access: Jan 2021.

[39] 2019. Espresso. https://developer.android.com/training/testing/espresso

[40] Android Debug Bridge (adb), Google, 2018. https://developer.android.com/studio/command-line/adb

[41] UI Automator, Google, 2020. https://developer.android.com/training/testing/ui-automator

[42] Android Emulator, Google, [Online]. Available: https://developer.android.com/studio/run/emulator

[43] GPT-4, Openai, 2023 Available: openai.com/gpt-4.

[44] Gemini, Google, 2023, Available: https://blog.google/technology/ai/google-gemini-ai/#performance

X. Experiences, Rewards, and Further Suggestions

The process of developing the test migration tool for test case migration between applications has been an exciting and challenging journey. From the outset, we were driven by the idea of creating a tool that would simplify and streamline the migration process for software testers. Understanding the needs of testers and envisioning a solution that could empower them to focus more on their core responsibilities added purpose to the project.

As development progressed, we remained focused on creating an intuitive and user-friendly tool. We wanted to ensure that testers could seamlessly transfer their test cases from one application to another without unnecessary complications. The challenges encountered along the way only fueled my commitment to continuous improvement.

Throughout the project, we encountered a lot of hardships. We had problems installing the necessary environment for the automation tools, but help from other teammates greatly relieved our issues. We also faced obstacles while developing our tool, and we sought help from our mentor. Admittedly, the research journey was not always as straightforward as 23 imagined. Code often seemed to defy our intentions, leaving us empty-handed at day's end. Yet, when breakthroughs occurred, my teammate and I celebrated as if we'd won a million-dollar lottery. Those "Eureka" moments, shrouded in mist and uncertainty, illuminated our pursuit of scientific research-problem solving. Like a detective following clues, we seek answers, driven by an insatiable hunger for understanding. It is even nicer than the problem we solve can contribute to something meaningful!

Beyond the research itself, our time in Shenzhen was enriched by the remarkable peers we encountered. These "geniuses" spanned diverse fields: some were already part of the national Olympic team, while others had discovered new species. Yet, their common thread was a humble dedication to work that mattered. Their passion and clash of ideas inspired me to delve deeper into specific topics, infusing my own pursuits with greater meaning. It was these companions, alongside the thrill of discovery, that made my Shenzhen experience truly delightful.

Overall, the program made a huge impact on our development. It offers a great opportunity for us to meet with some of the most prominent scholars in the country. The guest lectures and workshops from prominent researchers and scientists were another highlight of the camp. Their insights and experiences gave us a glimpse into the real-world challenges and opportunities in scientific research. Their passion and dedication inspired me to continue my pursuit of knowledge and contribute to the advancement of science.

As for the suggestions for future AXiS participants, we recommend finding that old saying- Work-Life Balance (WLB). Doing research is always exhausting, so it is important to find that balance between your normal life and academic life (really important for future college life). After all, research is not everything. Additionally, talk with your peers and professors. They are very knowledgeable and can help you to expand your perspective. At last, **have fun**.

Have Fun!