Intention-Based GUI Test Migration for Mobile Apps using Large Language Models

SHAOHENG CAO, Nanjing University, China
MINXUE PAN*, Nanjing University, China
YUANHONG LAN, Nanjing University, China
XUANDONG LI, Nanjing University, China

Graphical User Interface (GUI) testing is one of the primary quality assurance methods for mobile apps. Manually constructing high-quality test cases for GUI testing is costly and labor-intensive, leading to the development of various automated approaches that migrate test cases from a source app to a target app. Existing approaches predominantly treat this test migration task as a widget-matching problem, which performs well when the interaction logic between apps remains consistent. However, they struggle with variations in interaction logic for specific functionalities, a common scenario across different apps. To address this limitation, a novel approach named ITEM is introduced in this paper for the test migration task. Unlike existing works that model the problem as a widget-matching task, ITEM seeks a novel pathway by adopting a two-stage framework with the comprehension and reasoning capability of Large Language Models: first, a transition-aware mechanism for generating test intentions; and second, a dynamic reasoning-based mechanism for fulfilling these intentions. This approach maintains effectiveness regardless of variations across the source and target apps' interaction logic. Experimental results on 35 real-world Android apps across 280 test migration tasks demonstrate the superior effectiveness and efficiency of ITEM compared to state-of-the-art approaches.

CCS Concepts: • **Software and its engineering → Software testing and debugging**.

Additional Key Words and Phrases: Android Testing, GUI Test Migration

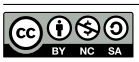
ACM Reference Format:
Shaoheng Cao, Minxue Pan, Yuanhong Lan, and Xuandong Li. 2025. Intention-Based GUI Test Migration for Mobile Apps using Large Language Models. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA101 (July 2025), 23 pages. https://doi.org/10.1145/3728978

1 Introduction

The rapid development of mobile apps has made them increasingly prevalent in all aspects of daily life. Recent data indicates that the Google Play Store hosts over 2.4 million Android apps, with approximately 30,000 new apps emerging weekly [2]. In this highly competitive market, maintaining app quality has become a critical priority for developers. Due to the GUI-driven characteristic of mobile apps, GUI testing has been adopted as one of the primary testing methods for quality assurance. A typical GUI test case used in GUI testing is a sequence of test actions and there are two primary types of test actions [3, 16, 33]: (1) *GUI events* that interact with the

*Corresponding author.

Authors' Contact Information: Shaoheng Cao, shaohengcao@smail.nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China; Minxue Pan, mxp@nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China; Yuanhong Lan, yhlan@smail.nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China; Xuandong Li, lxd@nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China.



ACM 2994-970X/2025/7-ARTISSTA101
https://doi.org/10.1145/3728978

app, such as clicks or swipes, and (2) *GUI assertions* that verify whether the tested functionalities meet the developers' expectation, primarily by confirming the presence or absence of specific widgets. To execute these test cases automatically, they are often written as test scripts in code and run using testing engines like Appium [1]. Given high-quality test scripts, GUI testing is widely practiced and demonstrated effective [13, 25]. However, manually constructing these scripts is both time-consuming and labor-intensive [11]. Consequently, various approaches have been proposed to migrate test scripts from a source app to an app under test (target app) with similar functionalities to reduce manual labor. Current research such as CraftDroid [16] and TEMdroid [33] predominantly models test migration as a widget-matching problem. These approaches establish the matching relation between widgets of the source and target apps, enabling the migration of test actions based on their associated widgets.

However, this framework has several limitations. Firstly, simply modeling test migration as widget matching fails to comprehensively address the migration challenge. Although current approaches perform well when apps share similar interaction logic for specific functionalities, they often become ineffective when there are minor or major differences in the interaction logic between apps. These inherent differences can lead to variations in the number of test actions required for testing the same functionality across different apps, complicating the migration process. Additionally, current approaches directly migrate widget-irrelevant test actions, such as back or swipe operations, without considering the outcomes of their execution, compromising migration accuracy.

To tackle these limitations, we propose a novel approach named ITeM (**I**ntention-preserving **Te**st **M**igration). Our key insight lies in that the essence of the test migration task is to perform the same test intentions across different apps. Accordingly, ITeM extracts the test intentions from the test script on the source app and then fulfills these intentions on the target app, thus completing the migration of test scripts. This new framework offers a distinctive pathway for effective test migration, focusing on the underlying intentions rather than mere widgets.

However, both extracting test intentions from the source test scripts and fulfilling them on the target app pose significant challenges, involving deep comprehension of the test scripts and the interaction logic of apps. Leveraging the advanced capabilities of Large Language Models (LLMs) in comprehension and logical reasoning [23], this paper develops sophisticated prompt engineering techniques to tackle these challenges. Specifically, ITeM begins by running the source test scripts on the source app to record its execution trace. This trace captures GUI context information before and after each test action, along with action attributes. We use this detailed information to construct transition-aware prompts for the LLM, enabling it to generate accurate test intentions.

After acquiring the test intentions of the source test script, ITeM then prompts the LLM to fulfill these intentions on the target app. As the required number of test actions for the same intention varies across apps, we have designed a dynamic reasoning-based framework to address this variability. This framework involves launching the target app on a device, dynamically prompting the LLM with each intention and the real-time GUI context of the app. The prompts are specially designed for the LLM to generate thoughts and actions [30]. The thoughts are analyzed to determine if the current intention can be fulfilled, requires further actions, or is unnecessary in the target app. Actions are parsed into events and dynamically executed on the device. Through this novel framework, ITeM effectively handles the interaction logic difference between apps, yielding promising migration results.

To evaluate the effectiveness and efficiency of ITeM, we applied it to 35 real-world Android apps across seven categories, encompassing a total of 280 migration tasks. The experimental results indicate that ITeM is highly effective, outperforming state-of-the-art (SOTA) approaches like CraftDroid [16] and TreaDroid [18]. Specifically, ITeM increased the average F1-score by 31% and the overall migration success rate by 26% compared to the SOTA approaches. Moreover, ITeM

proved to be highly efficient, with an overall migration time of 392.51 minutes for all 280 migration tasks.

The contributions of our work are as follows:

- We present ITeM, a novel approach that first leverages LLMs for test migration. The approach captures the essence of the test migration task, offering a comprehensive solution.
- We implement a supporting tool of ITeM, promoting the easy application and future research of test migration. The tool and dataset are publicly available[1].
- We evaluate ITeM's effectiveness and efficiency by applying it to conduct 280 migration tasks on 35 real-world Android apps. The results clearly indicate that ITeM outperforms existing approaches significantly.

2 Background

2.1 Test Intention

The term **test intention** in this paper refers to the tester's objective when writing the source test script; to be specific, the events they aim to trigger on the app and the expected behaviors when the script is executed. In this paper, we define each test intention as corresponding to a single test action in the source test script. When performing a test intention on the target app, it may require one or multiple test actions. Additionally, due to differences in interaction logic between the source and target apps, some test intentions may be unfulfillable on the target app.

The test intention is described in natural language, outlining the purpose of the corresponding test action. For example, *Click the "Sign In" button to Log In* represents a GUI event, while *Check the existence of the "Sign In" button* describes a GUI assertion. Generally, a test intention includes two basic elements: the operation description (e.g., *Click the "Sign In" button*) and the purpose of the operation (e.g., *to Log In*). The purpose is optional and can be merged with the description (e.g., for an assertion, *Check the existence of the "Sign In" button* encapsulates both description and purpose). This aligns with how humans naturally describe test actions.

Using natural language for test intentions provides several advantages. First, natural language is inherently flexible, making it easier for testers to understand and modify with minimal technical knowledge [14]. Second, original test scripts are typically app-specific and contain specialized IDs or text of the source app. This may compromise the potential for successful test migration. In contrast, using natural language can abstract away app-specific details, making the test intentions more general and enhancing the migration effectiveness—an advantage that will be highlighted in 4.3.4.

2.2 Motivating Example

Figure 1 shows a motivating example of our work. Specifically, GUI screens S1 and S2 belong to the "K-9 Mail" app (App1), while S3, S4, and S5 are from the "myMail" app (App2). The test intentions to be fulfilled are (1) *navigating to the sent folder*, (2) *check for the existence of the testing email*. In App1, the intentions are achieved by clicking the sent button (w1) to display the specific email (w2) on the screen, leading to the test case *click w1, check w2* (TC1). Conversely, in App2, fulfilling the same intentions involves expanding the sidebar by clicking w3 in S3, followed by clicking w4 and checking the existence of w5, resulting in the actions *click w3, click w4, check w5* (TC2). Migrating TC1 to App2 necessitates an additional action to expand the sidebar, whereas migrating TC2 to App1 requires omitting the redundant *click w3* action.

These two scenarios cannot be solved by widget matching because they both require adaptation to the number of test actions, not the one-to-one match formulated by widget matching. Some

[1] https://github.com/Vallxy/ITeM

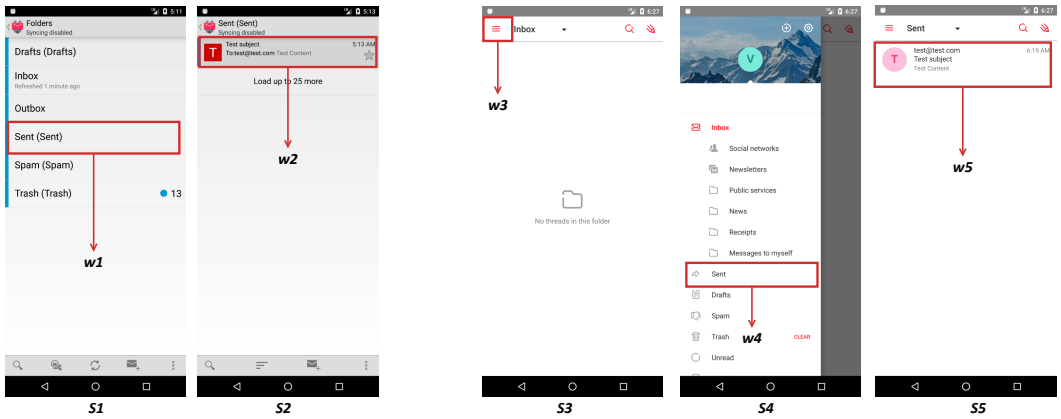


Fig. 1. GUI screens of two mailing apps for the opening up "sent" folder functionality

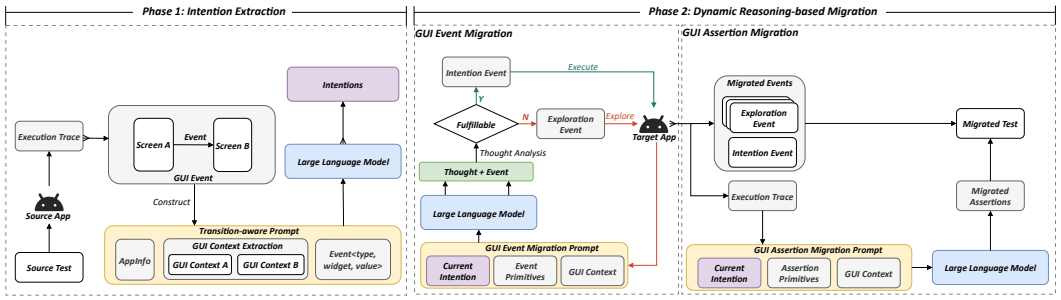


Fig. 2. Overview of ITEM

existing works ignore these scenarios, limiting their effectiveness. Others utilize transition graphs acquired from static analysis to assist in such scenarios. While this may be effective for some open-source and small-scale apps, it struggles with handling complex apps. Moreover, even with static analysis, these approaches fail to handle the migration task in this example. In contrast, our approach ITEM successfully manages bi-directional migration in this example. Its details will be elaborated upon in subsequent sections.

3 Approach

3.1 Overview

We present ITEM, a novel approach that leverages the LLM for test migration. Figure 2 illustrates the overview of our approach. The input of ITEM is the source test script written for the source app and the output is the migrated test script for the target app. The workflow of ITEM encompasses two main phases: (1) the *Intention Extraction*, which involves executing the source test script on the source app, recording the execution trace, and constructing transition-aware prompts for the LLM to generate test intentions for each test action of the source test script. (2) the *Dynamic Reasoning-based Migration*, which dynamically constructs prompts for the LLM to migrate GUI events and assertions. Specifically, it first migrates the GUI events and then the assertions. This order is chosen because migrating GUI events requires more exploration, which can involve backtracking and app recovery. In contrast, migrating GUI assertions typically does not require such exploration and largely depends on the successful migration of GUI events to reach the correct GUI states.

Table 1. Prompt Pattern for Test Intention Extraction

PROMPT	INFORMATION USAGE	EXAMPLE
Requirement	<*Task Description*>	I have an execution trace of a test case for an Android app. Your task is to analyze the trace and identify the test intention behind each test action of the trace.
App Information	<*Package Name*>, <*Activity Name*>	The basic information of the app before performing the test action includes: {current package: "com.google.example", current activity: "MainActivity"}.
GUI Context	<*GUI Context before Execution s*>, <*GUI Context after Execution s′*>	The GUI screen before performing the test action contains these widgets: [{resource-id: "org.mozilla.focus:id/search", class: "android.widget.Button", content-desc: "Search", text: "Skip", index: 0},...]. The GUI screen after performing ...
Action Information	<*Action Type t*>, <*Associated Widget w*>, <*Associated Data v*>	The information of the executed test action includes: {actiont type: "input", value: "www.google.com"}. The widget associated with the test action is: {resource-id: "org.mozilla.focus:id/search", class: "android.widget.Button", content-desc: "Search", text: "Search"}.
Output	<*Output Pattern*>	Your response should be <Intent: {Your answer}>e.g. <Intent: click the "Sign In" button to log in>.
Example output	<Intent: input (www.google.com) into the search bar and navigate to the website>	

3.2 Intention Extraction

The *Intention Extraction* phase aims to extract the test intention for each test action of the source test script. These test intentions will be fulfilled on the target app to complete the test migration task. The extraction process involves understanding the execution trace and summarizing the test intentions from the trace. We leverage LLMs for this task due to their strong comprehension and generation capabilities empowered by pre-training on massive data. As illustrated in Figure 2, this phase starts by executing the source test script on the source app. The execution trace, denoted as $\mathcal{T}$, is a set of transitions. Each transition $\tau \in \mathcal{T}$ is a five-tuple $\langle s, t, w, v, s' \rangle$, where s and s' represent the GUI context before and after executing the test action. The w and v represent the associated widget and the data used for the test action, respectively, while t represents the type of the test action. Specifically, w includes all attributes of the widget, such as *resource-id*, *content-desc*, and *text*. If the test action is widget-irrelevant, w is set to *null*. The GUI context refers to the list of widgets present in the GUI at a specific time. In Android, each GUI screen of an app can be captured and parsed into an XML file using the *uiautomatorviewer* provided by the Android SDK. We extract the widgets through this tool. To filter irrelevant widgets and reduce the length of the prompt, we remove layout widgets and widgets with empty attributes.

After acquiring the trace of the source test, transition-aware prompts are constructed for the LLM to generate test intentions. Table 1 presents the prompt pattern and an output example of this phase. We now elaborate on the details of the prompt pattern.

App Information. The app information includes essential details about the runtime application, i.e. the package name and activity name. These pieces of information often contain keywords that describe the app's functionality. For example, "com.k9.mail" indicates the app is an email application, and "AccountSettingActivity" suggests the current GUI screen is related to settings functionality.

GUI Context. For the GUI context, instead of merely considering the GUI state before execution, we leverage both the GUI state before and after the test action execution. This approach better reflects the transition caused by the test action, enabling a more precise generation of test

intentions. Additionally, we filter out unnecessary widgets in the GUI context, such as layouts (e.g., RelativeLayout) and empty widgets, which are not needed for our task and will bring about extensive token costs.

Action Information. The action information specifies the details of the test action, including the action type, the widget involved, and the data used. This information is crucial for determining the test intention. Our approach currently supports the same action types as SOTA works, including *click*, *long click*, *swipe*, *scroll*, *back*, *input*, and *enter* for GUI events, as well as *widget-relevant check* and *widget-irrelevant check* for assertions, which are the most commonly used in test scripts [16]. Other action types can also be easily supported since our work is not restricted to specific actions.

Output format. We designed an instruction to guide the output format of the LLM. In this instruction, we do not constrain the output to a structured format, instead, we choose to adopt the natural language form. Our experimental results, which can be found in Section 4.3.4, will demonstrate the superiority of natural language intentions over structured formats. To make the generated test intentions contain the essential elements as required in Section 2.1, we also incorporate example outputs within the instruction, leveraging the concept of few-shot learning to improve the robustness of the output.

3.3 Dynamic Reasoning-Based Migration

In this phase, test intentions extracted from the source test scripts are utilized to guide the migration of tests to the target app. The main challenge lies in the uncertainty of the number of test actions needed to fulfill the intentions on the target app, due to variations in GUI appearance, widgets, and interaction logic between the source and target app. This necessitates a reasoning process to either eliminate redundant test intentions or construct additional test actions, a task that is non-trivial even for experienced human testers. To effectively tackle these challenges, we have developed comprehensive prompts based on the ReAct prompt framework [30]. The *Dynamic Reasoning-based Migration* phase can be divided into two key stages: *GUI Event Migration* and *GUI Assertion Migration*.

3.3.1 GUI Event Migration. Migrating GUI events is notably more complex than migrating assertions as it involves the transition of GUI states, which faces two main challenges:

- C1. How do we construct new GUI events that accurately fulfill a specific test intention?
- C2. How can we confirm that a test intention has been successfully migrated so that the migration process can be carried on without influencing the subsequent ones?

To address C1, we must determine the number of new GUI events required to execute a test intention on the target app. For example, if a menu item in the source app lies in a submenu in the target app, accessing it requires an additional click. This indicates a more complex GUI event sequence than in the source app. Conversely, if no GUI events are needed, it may suggest that the particular intention is unnecessary in the context of the target app and should be omitted. Our approach involves iteratively prompting the LLM based on the real-time GUI of the target app and analyzing the LLM's feedback to construct the new GUI events. This process continues until a test intention is either fully fulfilled or deemed unnecessary.

The detailed process is described in Algorithm 1. For each intention to be fulfilled, the algorithm initializes a step limit to a pre-defined threshold (lines 2-3), whose influence will be discussed in Section 4.3.4. Within the step limit, the algorithm prompts the LLM for possible GUI events to fulfill the intention. It dynamically captures the GUI context and constructs GUI migration prompts for the LLM (lines 6-7). The outcome of the LLM contains two parts: the thought and event (line 8). The thought is a tag generated by the LLM to determine the relationship between the generated event and the intention to be fulfilled. If the LLM tags the generated event with "Explore", it indicates that

Algorithm 1: GUI Event Migration

Input: Exploration Step Threshold θ, Test Intentions $\mathcal{I}$
Output: Migrated GUI Events Sequence $\mathcal{E}$

```
1  E ← [];
2  foreach i ∈ I do
3      stepLimit ← θ;
4      E_exploration ← [];
5      while stepLimit > 0 do
6          s ← captureGUIContext();
7          prompt ← constructPrompt(s, i, E_exploration);
8          thought, event ← promptLLM(prompt);
9          if thought == "Explore" then
10             E_exploration.append(event);
11             execute(event);
12         end
13         else if thought == "Intention" then
14             E.append(all events ∈ E_exploration);
15             E.append(event);
16             execute(event);
17             break;
18         end
19         stepLimit ← stepLimit − 1;
20     end
21     if stepLimit == 0 then
22         recover(E);              ▷ Reset the app and re-execute all the migrated events ∈ E.
23     end
24 end
25 return E;
```

executing this event alone does not fully fulfill the current intention but is part of an exploratory process. The event is then executed and recorded in the exploration event list for the current test intention (lines 9-12). If the LLM tags its generated event as "Intention", it indicates that after the execution of this event, the current test intention would be fully fulfilled. Consequently, the exploration event list $\mathcal{E}_{exploration}$, followed by this newly generated intention event, are considered the event list that fulfills the current intention and is added to the final event list (lines 13-18). This design supports scenarios where multiple events are needed to fulfill a single test intention. If no intention event is generated within the step limit, the intention is considered unnecessary for the target app and is skipped. In these cases, the app is recovered to its previous state by resetting the app and re-executing the previously generated GUI events, then proceeding with the next intention (lines 21-23).

This algorithm effectively addresses C1 by prompting the LLM to produce reasonable events or determine skipping them. Additionally, it also addresses C2, as the generation of the event which is tagged as *Intention* indicates the completeness of the current intention. Experimental results in 4.3.4 demonstrate the superior effectiveness and efficiency of this algorithm.

Table 2 shows the pattern of the prompt used in the GUI event migration algorithm. We now elaborate on its details.

GUI Context. Since the prompting process is dynamic, the GUI context includes the current widgets on the GUI screen, allowing the LLM to understand the GUI state. It also contains a list of interactive widgets for the LLM to choose from when performing the GUI event.

Table 2. Prompt Pattern for GUI Event Migration

PROMPT	INFORMATION USAGE	EXAMPLE
Requirement	<*Task Description*>, <*Test Intention*>	Determine the necessary operation to fulfill the test intention "input (www.google.com) into the search bar and navigate to the website".
GUI Context	<*current GUI context*>, <*Interactive Widget Indexes*>	The current screen contains some widgets, listed as: [{resource-id: "acr.browser.lightning:id/search", class: "android.widget.EditText", content-desc: "", text: "Search"},...], of which the ith (i in [0,1,2,4,...]) is interactive.
Event Types	<*Supported Event Types*>	Available operations include: [click, long click, back, input, swipe right, swipe left, scroll up, scroll down]
Reasoning	<*Reasoning Activation Prompt*>	If the intention cannot be achieved in a single step, feel free to explore the application further. If you decide to explore the app to reach the functionality, please tag your response with "Explore"; otherwise, use "Intention".
Exploration History	<*Explored GUI Event List* >	You were exploring the app. The exploration history contains: [<Explore, click, w1>, <Explore, back, null>, ...]
Output	<*Output Pattern*>	Please limit your answer to one operation per response in the format <{Explore/Intention}, {Event Type}, {Widget Index}, {Input Value}>. e.g.<Explore, input, 3, test@gmail.com>
Example output	<Intention, input, 0, "www.google.com">	

Event Type. To facilitate the production of the events for execution on the target app, we provide a list of supported GUI events for the LLM to select from.

Reasoning. The reasoning activation prompt triggers the LLM to start a reasoning process about its generated event [30], determining whether to explore the app to navigate to the correct GUI state where the intention can be performed. This enhances the LLM's reasoning capability, contributing to the success of the GUI event migration. This is also inspired by works that prompt LLMs to interact with external information and tools [12, 29]. Additionally, this design introduces a self-validation process [15] for the generated events, enhancing the consistency of the LLM's behavior and reducing the randomness of its outputs. If the LLM decides to explore, the prompt will include the exploration history, as illustrated in the table.

Output Format. The output contains a thought and a GUI event. The thought is represented as a tag "Explore" or "Intention", indicating the LLM's assessment of whether executing the current GUI event could fully complete the input test intention. The GUI event consists of the event type, widget index, and the data used. Specifically, when representing a widget, rather than letting the LLM replicate an entire widget with all its attributes, we adopt a more effective method: sequentially encoding each widget to assign indices, enabling the LLM to accurately represent a widget by its index. This method has been proven to generate more consistent and accurate outputs, as supported by research [6, 9]. We apply this encoding technique across all prompts within our approach.

3.3.2 GUI Assertion Migration. After the *GUI Event Migration* stage, the migrated events are generated along with an execution trace. Since assertions do not involve operations that change the GUI state, we exclude the migration of assertions in the prior stage to reduce the time and prompt token costs caused by exploration. With the execution trace and the intentions, we can locate the

Table 3. Prompt Pattern for GUI Assertion Migration

PROMPT	INFORMATION USAGE	EXAMPLE
Requirement	<*Task Description*>	Assist in transferring a test assertion from one app to another based on the provided details. Identify a corresponding element in the new GUI that fulfills a similar role.
Assertion	<*Source Widget*>, <*Source GUI Context*>, <*Target GUI Context*>	This source widget is identified by {resource-id: "acr.browser.lightning:id/search", class: "android.widget.EditText", content-desc: "", text: "Search"} within a GUI containing: [{resource-id: "acr.browser.lightning:id/search", class: "android.widget.EditText", content-desc: "", text: "Search"},...]. The new GUI contains: [{},...]
Output	<*Output Pattern*>	Provide the index of the matching element from the new widget list. The index starts from 0. If there is no matching widget, return -1. The format should be <index:>, e.g. <index:0>, <index:-1>
Example output	<index:0>	

assertion to find its associated GUI context. Our approach supports two categories of assertions, as do SOTA approaches: widget-irrelevant and widget-relevant existence/non-existence checking.

For widget-irrelevant assertions, direct migration is possible as long as its preceding GUI event has been migrated correctly. Consequently, we adopt a straightforward approach to migrate these assertions directly. For widget-relevant assertions, the correct migration of preceding GUI events simplifies the assertion migration problem to finding the matching widget, a widget-matching problem. Previous works leverage widget attributes and employ pre-trained or trained models to measure the similarity of these attributes, thereby determining the matching relationship of widgets. Instead of directly applying these approaches, we explore whether the LLM can handle widget-matching given the same attributes, leveraging its summary and language understanding capabilities. We select three attributes that are universally used in widget-matching migration approaches: *resource-id*, *content-desc*, and *text*. Note that we do not explore LLM for widget matching in the GUI event migration, as this problem extends beyond simple widget matching.

Specifically, we carefully design the prompts for GUI assertion migration. The detailed prompt pattern is shown in Table 3. The prompt includes the widget information, the source GUI context where the source assertion is performed, and the target app's GUI context where the migrated assertion is desired to be performed.

3.4 Other Prompt Design

Designing prompts for the LLM to get desired outcomes is a non-trivial task [35]. Various universal prompt techniques have been proposed to improve the quality and robustness of LLM's outcomes.

Chain-of-Thought (CoT) is a prompt technique that enables an LLM to reason in a stepwise manner, thereby enhancing its reasoning capabilities [5, 26]. It is implemented by including the phrase "*Let's think step by step.*" within the prompt. In our approach, CoT has been systematically applied across all prompts to optimize the LLM's performance.

Few-shot in-context learning is a prompt technique that provides a small number of examples to the LLM, helping it understand the reasoning process and thereby producing more convincing results [24]. In our approach, this technique is applied to the output format prompt to enhance the robustness of it. Without this technique, the output may occasionally contain some unexpected or unrelated content that is significantly different from what we intend.

Table 4. Experimental Subjects

Category	App	Version	Functionality	#TC	#E	#A
a1-Browser	a11-Lightning	4.5.1	b11-Access website by URL	5	12	5
	a12-Browser for Android	6.0				
	a13-Privacy Browser	2.10				
	a14-FOSS Browser	5.8	b12-Back navigation	5	22	15
	a15-Firefox Focus	6.0				
a2-To Do List	a21-Minimal	1.2	b21-Add an item to list	5	15	5
	a22-Clear List	1.5.6				
	a23-To-Do List	2.1				
	a24-Simply Do	0.9.1	b22-Remove list item	5	23	10
	a25-Shopping List	0.10.1				
a3-Shopping	a31-Geek	2.3.7	b31-Registration	5	37	12
	a32-Wish	4.22.6				
	a33-Rainbow Shops	1.2.9				
	a34-Esty	5.6.0	b32-Login	5	21	11
	a35-Yelp	10.21.1				
a4-Mail	a41-K-9 Mail	5.403	b41-Search email by keywords	5	10	15
	a42-Email mail box fast mail	1.12.20				
	a43-Mail.Ru	7.5.0				
	a44-myMail	7.5.0	b42-Send email	5	35	15
	a45-Gmail	5.5.10				
a5-Calculator	a51-Tip Calculator	1.1	a51-Calculate total bill with tip	5	11	5
	a52-Tip Calc	1.11				
	a53-Simple Tip Calculator	1.2				
	a54-Tip Calculator Plus	5.5.10	a52-Split bill	5	16	5
	a55-Free Tip Calculator	1.0.0.9				
a6-News	a61-ABC News	5.23.2	a61-Open settings to provide feedback	5	17	10
	a62-BuzzFeed	3.29.2				
	a63-SmartNews	5.10.0				
	a64-Google News	5.114	a62-Search news	5	17	5
	a65-Mail Online	5.15.1				
a7-Music Player	a71-Vinly Music Player	1.11.0	a71-Search music to play	5	30	15
	a72-APlayer	1.6.4.0				
	a73-Auxio	3.6.3				
	a74-Retro Music	6.2.1	a72-Add a new playlist	5	26	10
	a75-Phonograph Plus	1.9.1				

4 Evaluation

To evaluate the effectiveness and efficiency of ITeM, we conducted experiments that apply ITeM to migrate test scripts for mobile apps. Based on the experimental results, we address the following research questions:

- RQ1: How accurate is ITeM in generating test intentions?
- RQ2: How effective is ITeM in migrating GUI test scripts?
- RQ3: How efficient is ITeM in migrating GUI test scripts?
- RQ4: How do different components and configurations affect ITeM's performance?
- RQ5: How does ITeM perform on different LLMs?

4.1 Experimental Subjects

There are three primary datasets utilized for test migration: the CraftDroid, FrUITeR [36], and SemFinder [22] datasets. The CraftDroid dataset comprises 25 Android applications across five categories, with two test scripts per app to test different functionalities. One app in the "Mail" category was outdated and non-executable, so we substituted it with "Gmail" and manually composed equivalent test scripts. The FrUITeR dataset includes 20 apps across two categories, one of which overlaps with CraftDroid. For consistency, we retained the overlapping category from

CraftDroid. In FrUITeR's another category, "News", seven of ten apps were non-executable due to obsolescence, leading us to supplement the remaining three with "Google News" and "Mail Online" collected from the Google PlayStore to maintain parity with CraftDroid's category size. FrUITeR's "News" category initially provided 14 test cases containing event sequences without assertions; we randomly selected two and manually appended assertions. The SemFinder dataset was excluded due to its focus on widget matching pairs rather than complete test scripts. Additionally, we created a new "Music Player" category consisting of apps released after May 20, 2024, which is the latest knowledge cutoff date for all evaluated LLMs in our experiments, to create a category of likely unseen data. We recruited three graduate students in Computer Science, each with at least two years of experience in Android testing, to construct two test scripts for this category and add assertions for the "News" category. Each test script was independently authored by one student and validated by the other two to ensure correctness.

Table 4 presents the basic information about the subject apps. Each category includes two test scripts per app to test different functionalities. For each test script within a category, the migration task involves migrating the test script from one of the five apps (source) to the remaining four apps (target), resulting in 20 (5×4) migration tasks per category and 280 (20×14) migration tasks in total. The target test script serves as the reference for the ground truth. Table 4 also shows the details of the test scripts with #E and #A denoting the number of GUI events and assertions, respectively. We further categorized the migration tasks into three types of migrations: **1-1 migration**, which indicates that all the test actions in the source test script can find a matching action in the target ground truth test script, strictly following a one-to-one matching relation; **1-n migration**, which involves more test actions to fulfill the original test intentions on the target app; **n-1 migration**, where fewer test actions are needed to fulfill the test intentions on the target app. The 1-n and n-1 migrations derive from differences in interaction logic and the number of actions needed to fulfill a specific intention between the source and target apps. Among the 280 migration tasks, 47% (132/280) are 1-1 migrations while 1-n and n-1 migrations account for 27% (75/280) and 26% (73/280), respectively.

4.2 Experimental Protocol

The experiments were conducted on a Windows PC equipped with an AMD Ryzen 9 5950X CPU, 64GB RAM, and an RTX 3050 GPU. The subject Android apps were executed on Android emulators with API level 23 and 30. The default LLM used by ITeM is GPT-4-turbo.

To answer RQ1, we applied ITeM to generate test intentions for all source test scripts. The three graduate students involved in the dataset construction assessed whether the test intentions accurately reflected the source test actions. Each student was provided with the source test scripts, their execution traces, and the test intentions generated by ITeM. They were asked to assess whether each generated intention accurately captured the true test intention of the corresponding test action, with the criterion that the intentions could be correctly fulfilled on the source app manually. A test intention was deemed accurate only if all three students reached a consensus.

To answer RQ2 and RQ3, we applied ITeM to migrate test scripts for the subjects and compared its effectiveness and efficiency with SOTA test migration approaches: TEMDroid, TreaDroid [18] CraftDroid and AppTestMigrator (ATM). Unfortunately, the publicly available version of TEMdroid is not executable due to missing critical files. Our communication with the authors failed to resolve these implementation gaps. Therefore, we used the data presented in their paper for comparison. Their experimental dataset is a subset of ours, so our comparison with TEMdroid is limited to this subset. Additionally, we noted that some test repair approaches adopt a similar widget-matching framework. Test repair involves repairing obsolete test scripts to restore their original test intentions on newer versions of apps, which can also be seen as a form of test migration [4].

Therefore, we included GUIDER [27], one of the SOTA test repair approaches, as an additional baseline for comparison. To ensure a comprehensive evaluation, we also incorporated AdbGPT [9], a SOTA bug reproduction approach that leverages LLMs and shares similar objectives with our approach.

To answer RQ4, we configured the following experimental settings: (1) We modified ITEM to produce variants. Specifically, we removed the *Intention Extraction* process and used a direct combination of action type and widget information as input, a common structured format used in scripts [4, 16, 27]. This allowed us to investigate the benefit of using natural language form of intentions over the original structured format in the test scripts. Additionally, we replaced the *Dynamic Reasoning-based Migration* with simpler prompts by excluding the reasoning component in the prompt (see Table 2). This modification allowed us to assess the impact of the dynamic reasoning mechanism on the overall performance of the approach. (2) We adjusted the exploration step limit threshold θ and compared the effectiveness and efficiency of ITEM.

To answer RQ5, we evaluated the effectiveness of ITEM using different LLMs, including GPT-4, Llama 3.1 [8], and GLM-4 [31], representing both closed-source and open-source models.

In experiments, we adopt the metrics *Precision*, *Recall*, and *F1-score* used by previous works [3, 16, 33, 36] to measure the effectiveness. When calculating *Precision* and *Recall*, we need to use True Positive (TP), False Positive (FP), and False Negative (FN) metrics. A TP occurs when a source test action is migrated correctly to the ground-truth target test action(s); an FP occurs when a source test action is migrated to test action(s) that are not in the ground-truth test case; an FN occurs when a ground-truth test action is not fulfilled. Besides, we also recorded migration time cost to evaluate the efficiency. Since LLMs can return different answers to the same prompt, for the LLM-powered approaches in our experiments, we conducted three runs and recorded the average metrics. Additionally, because AdbGPT does not support assertion manipulation, we only recorded its performance on GUI events.

To evaluate the overall performance and practical utility of the migrated test scripts, we introduced the *Success Rate*, which represents the percentage of migrated scripts that successfully fulfill the original test intentions. The three students mentioned above were tasked with reviewing the migrated scripts and their execution traces, comparing them against the original scripts and execution traces to determine whether the migrated script fully fulfilled the original test intentions. A migration was considered successful only when all three students reached a consensus.

4.3 Experimental Results

4.3.1 RQ1-Accuracy. Among the 292 generated test intentions for GUI events and 138 for GUI assertions, all intentions for the GUI assertions were correct. Only 1% (3/292) of the test intentions for GUI events were marked as incorrect. Upon inspection, we found that these inaccuracies stemmed from the same test action across three apps in the *To Do List* category, specifically the action to open an options menu for a list item. For five apps in this category, three required a swipe action on the list item, while the other two required a long press. One of the three students marked these three corresponding test intentions as incorrect. These intentions specified *Open the options menu for "Sample todo" list item* but did not include instructions on the action type required. The student attempted a long press, which failed on the three swipe-based apps. The other two students deemed all test intentions correct. Notably, when we used ITEM to re-fulfill the source intentions on the source app, all intentions, including these three, were successfully fulfilled.

4.3.2 RQ2-Effectiveness. As presented in Table 5, our approach ITEM achieves superior results across all metrics and functionalities, including GUI events and GUI assertions. Specifically, ITEM shows an increase of 26%, 30%, and 31% in overall Precision, Recall, and F1-score compared with the

Table 5. Comparison between ITeM, CraftDroid, GUIDER, ATM, AdbGPT and TreaDroid

Functionality	Approach	GUI Event			GUI Assertion			All			T (min)
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	
b11,b12	CraftDroid	0.79	0.69	0.73	0.78	0.66	0.72	0.79	0.68	0.73	594.17
	GUIDER	0.39	0.65	0.49	0.43	0.58	0.49	0.40	0.62	0.49	66.13
	ATM	0.69	0.58	0.63	0.56	0.66	0.61	0.63	0.61	0.62	51.93
	AdbGPT	0.59	0.55	0.57	N/A	N/A	N/A	N/A	N/A	N/A	25.87
	TreaDroid	0.79	0.63	0.70	0.81	0.56	0.66	0.80	0.60	0.69	841.87
	ITeM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	36.10
b21,b22	CraftDroid	0.69	0.79	0.73	0.63	0.74	0.68	0.67	0.78	0.72	793.18
	GUIDER	0.22	0.44	0.29	0.24	0.42	0.31	0.23	0.44	0.30	61.34
	ATM	0.41	0.39	0.40	0.41	0.32	0.36	0.41	0.37	0.39	12.25
	AdbGPT	0.49	0.28	0.35	N/A	N/A	N/A	N/A	N/A	N/A	22.12
	TreaDroid	0.63	0.52	0.57	0.57	0.27	0.37	0.62	0.45	0.52	1340.63
	ITeM	**0.88**	**0.86**	**0.87**	**0.93**	**1.00**	**0.96**	**0.90**	**0.90**	**0.90**	27.56
b31,b32	CraftDroid	0.38	0.76	0.51	0.36	0.57	0.44	0.38	0.70	0.49	573.05
	GUIDER	0.19	0.51	0.28	0.17	0.30	0.21	0.19	0.44	0.26	16.94
	ATM	0.22	0.42	0.29	0.45	0.48	0.46	0.27	0.44	0.34	107.44
	AdbGPT	0.12	0.03	0.05	N/A	N/A	N/A	N/A	N/A	N/A	21.63
	TreaDroid	0.39	0.26	0.31	0.29	0.19	0.23	0.36	0.24	0.29	349.20
	ITeM	**0.88**	**0.94**	**0.91**	**0.84**	**0.90**	**0.87**	**0.87**	**0.93**	**0.90**	70.85
b41,b42	CraftDroid	0.81	0.79	0.80	0.88	0.87	0.87	0.84	0.83	0.83	66.68
	GUIDER	0.75	0.68	0.71	0.84	0.76	0.80	0.79	0.71	0.75	40.53
	ATM	0.65	0.79	0.71	0.78	0.82	0.80	0.70	0.80	0.75	42.92
	AdbGPT	0.61	0.22	0.22	N/A	N/A	N/A	N/A	N/A	N/A	17.36
	TreaDroid	0.86	0.81	0.83	0.88	0.81	0.84	0.87	0.81	0.83	89.69
	ITeM	**0.99**	**1.00**	**0.99**	**0.99**	**1.00**	**1.00**	**0.99**	**1.00**	**0.99**	21.85
b51,b52	CraftDroid	0.80	1.00	0.89	1.00	0.73	0.84	0.84	0.91	0.88	212.67
	GUIDER	0.28	0.40	0.33	0.07	0.13	0.09	0.22	0.33	0.27	39.17
	ATM	0.48	0.65	0.55	0.32	0.53	0.40	0.43	0.62	0.51	32.76
	AdbGPT	0.48	0.39	0.43	N/A	N/A	N/A	N/A	N/A	N/A	19.44
	TreaDroid	0.86	0.83	0.85	0.82	0.78	0.79	0.85	0.82	0.83	503.03
	ITeM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	11.47
b61,b62	CraftDroid	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	842.54
	GUIDER	0.17	0.17	0.17	0.22	0.22	0.22	0.18	0.19	0.19	37.01
	ATM	0.14	0.13	0.14	0.23	0.23	0.23	0.17	0.16	0.17	26.45
	AdbGPT	0.45	0.31	0.36	N/A	N/A	N/A	N/A	N/A	N/A	23.25
	TreaDroid	0.25	0.16	0.20	0.06	0.03	0.04	0.20	0.12	0.15	1237.31
	ITeM	**0.65**	**0.51**	**0.57**	**0.35**	**0.35**	**0.35**	**0.54**	**0.46**	**0.49**	38.12
b71,b72	CraftDroid	0.22	0.15	0.18	0.43	0.28	0.34	0.28	0.19	0.23	474.87
	GUIDER	0.26	0.26	0.26	0.22	0.22	0.22	0.25	0.25	0.25	203.04
	ATM	0.15	0.15	0.15	0.09	0.09	0.09	0.13	0.13	0.13	133.63
	AdbGPT	0.35	0.13	0.19	N/A	N/A	N/A	N/A	N/A	N/A	22.52
	TreaDroid	0.45	0.26	0.33	0.43	0.25	0.32	0.45	0.25	0.32	642.57
	ITeM	**0.72**	**0.76**	**0.74**	**0.76**	**0.74**	**0.75**	**0.73**	**0.76**	**0.74**	188.59
Overall	CraftDroid	0.52	0.55	0.53	0.59	0.57	0.58	0.54	0.56	0.55	3557.17
	GUIDER	0.31	0.41	0.35	0.36	0.42	0.39	0.33	0.41	0.37	584.65
	ATM	0.35	0.39	0.37	0.42	0.44	0.43	0.37	0.41	0.39	407.39
	AdbGPT	0.46	0.24	0.32	N/A	N/A	N/A	N/A	N/A	N/A	152.19
	TreaDroid	0.61	0.46	0.52	0.58	0.39	0.47	0.60	0.44	0.51	5004.29
	ITeM	**0.87**	**0.87**	**0.87**	**0.85**	**0.86**	**0.85**	**0.86**	**0.86**	**0.86**	392.51

best-performing baseline, respectively. Furthermore, as for each functionality, Table 5 also shows that ITeM always achieves superior results on each metric than the other baselines. Additionally, for the potentially unseen data (a7-Music Player) by the LLM, ITeM still achieves promising results, highlighting its generalizability. Similarly, Table 6 highlights ITeM's superior effectiveness, with improvements of 18%, 5%, and 13% in Precision, Recall, and F1-score compared with TEMdroid, respectively.

As to the success rate of migrated scripts, Table 7 shows that ITeM achieves the highest success rate across all the functionalities. As to the overall success rate, it achieves 54% (150/280), which demonstrates an increase of 28%, 39%, 41%, 26% compared with CraftDroid, GUIDER, ATM, and TreaDroid, respectively. This demonstrates the superior usefulness of the migrated scripts produced by our approach.

From these data, it is robustly proven that ITeM is more effective than the compared SOTA approaches. This improvement is due to ITeM's superior ability to handle interaction logic differences between the source and target apps.

Although ITeM has achieved promising results across all metrics, there are still cases it cannot handle correctly. We inspected all the migration results produced by ITeM to understand the

Table 6. Comparison between ITeM and TEMdroid

		GUI Event			GUI Assertion			All		
Functionality	Approach	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
b11,b12	TEMdroid	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	ITeM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
b21,b22	TEMdroid	0.76	**0.90**	0.82	0.70	**1.00**	0.82	0.74	**0.93**	0.82
	ITeM	**0.88**	0.86	**0.87**	**0.93**	**1.00**	**0.96**	**0.90**	0.90	**0.90**
b31,b32	TEMdroid	0.75	0.82	0.78	0.83	0.71	0.76	0.76	0.80	0.78
	ITeM	**0.88**	**0.94**	**0.91**	**0.84**	**0.90**	**0.87**	**0.87**	**0.93**	**0.90**
b41,b42	TEMdroid	0.88	**1.00**	0.94	0.88	**1.00**	0.94	0.88	**1.00**	0.94
	ITeM	**0.98**	**1.00**	**0.99**	**0.99**	**1.00**	**0.99**	**0.98**	**1.00**	**0.99**
b51,b52	TEMdroid	0.87	0.90	0.88	**1.00**	**1.00**	**1.00**	0.90	0.92	0.91
	ITeM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Overall	TEMdroid	0.71	0.93	0.81	0.90	0.89	0.90	0.77	0.91	0.83
	ITeM	**0.94**	**0.96**	**0.95**	**0.95**	**0.98**	**0.97**	**0.95**	**0.96**	**0.96**

Table 7. Comparison of success rate

	Functionality														
Approach	b11	b12	b21	b22	b31	b32	b41	b42	b51	b52	b61	b62	b71	b72	Total
CraftDroid	0.45	0.40	0.45	0.00	0.00	0.05	**1.00**	0.00	0.60	0.65	0.00	0.00	0.00	0.00	0.26
GUIDER	0.40	0.30	0.20	0.00	**0.10**	0.00	**1.00**	0.00	0.05	0.05	0.00	0.00	0.00	0.00	0.15
ATM	0.25	0.25	0.15	0.00	0.00	0.00	**1.00**	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.13
AdbGPT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TreaDroid	0.55	0.40	0.45	0.00	**0.10**	0.00	**1.00**	0.00	0.65	0.70	0.00	0.00	0.10	0.00	0.28
ITeM	**1.00**	**1.00**	**0.55**	**0.35**	**0.10**	**0.35**	**1.00**	**0.85**	**1.00**	**1.00**	0.00	0.00	**0.15**	**0.15**	**0.54**

reasons behind its ineffectiveness, particularly in producing FPs and FNs, which decrease Precision and Recall, respectively.

For GUI events, the primary reason for FPs is that the event produced by the LLM is incorrect for fulfilling the corresponding test intention. The errors stem from selecting the wrong event type or widget, with the latter being more common. For instance, several interactive widgets may have similar attributes, but only one serves the intended functionality. In such cases, the LLM may choose the wrong one. However, this issue can also be challenging for human testers who rely solely on GUI information to distinguish those widgets. Developing an extensive exploration mechanism for further confirmation of the generated event is a potential improvement we plan to explore in the future. For FNs, the main reason lies in the variation of event types. For example, removing an item from a list can vary across apps—some may involve swiping right on the item, while others may require a long click followed by selecting "delete" from a popup menu. While the LLM performs well in understanding and producing normal event types, it struggles with such scenarios. Even human testers may find it difficult to determine the correct event type without trial or source code information. Incorporating additional information from static analysis may help solve this problem, and we will explore this approach in the future.

For GUI assertions, the production of FPs and FNs both stem from the FPs and FNs in migrated GUI events. FPs and FNs in GUI events will navigate the target app to incorrect GUI screens. These incorrect GUI screens may not contain the corresponding widgets for migrating GUI assertions, leading to FPs and FNs in GUI assertions.

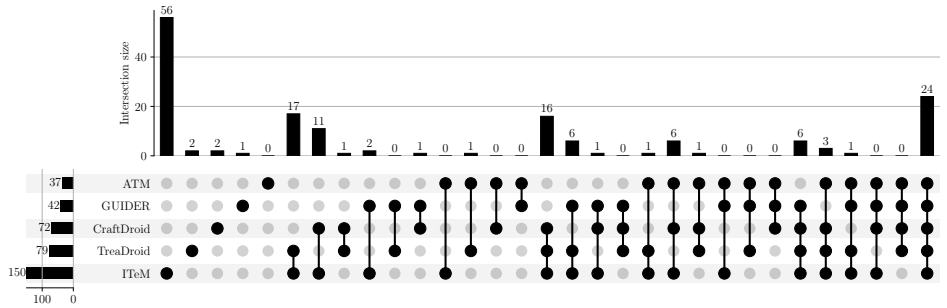


Fig. 3. Partition of the successfully migrated test scripts produced by GUIDER, ATM, CraftDroid, TreaDroid, and ITeM. Each column within the plot indicates the number of test scripts that are successfully migrated with the intention completely preserved by the approaches denoted by black dots but failed to be migrated by the approaches denoted by grey dots.

We examined all migrated scripts produced by each approach to analyze the differences in their effectiveness. Since AdbGPT could not migrate assertions, thus producing a zero success rate, we exclude it in this analysis. As is shown in Figure 3, only 9% (24/280) of migrated test scripts successfully preserved the original test intentions across all approaches. ITeM uniquely migrated 56 test scripts that none of the other approaches could handle, accounting for 20% of all migration tasks. We examined these unique migrations and found that 46% (26/56) of them involved 1-n or n-1 migrations. ITeM's dynamic reasoning mechanism enables it to handle situations where the number of test actions requires adjustment, facilitating these more successful 1-n and n-1 migrations.

Additionally, 54% (30/56) of the unique migrations were 1-1 migrations that can only be mitigated by ITeM. After examining the incorrect migrations from the baseline approaches, we found that the primary cause of failure was inaccurate widget matching, often due to a significant semantic gap between source and target widgets. This mismatch led to incorrect actions, failing to fulfill specific test intentions and subsequent ones. In contrast, ITeM accurately identified matching widgets in these 1-1 migrations due to its intention extraction component, which generalized test intentions effectively, and the LLM's comprehension ability, which helped understand the GUI state and locate corresponding widgets accurately.

However, Figure 3 also shows that some test scripts were successfully migrated by one or more approaches other than ITeM. GUIDER successfully migrated one unique test script that other approaches could not. GUIDER combines both widget attributes and visual information for widget matching, whereas the other approaches rely solely on attributes. In this case, the other approaches failed due to incorrect widget matching based solely on attributes, while GUIDER accurately utilized visual information for correct matching. Although incorporating visual information in ITeM might improve this case, it would increase token costs significantly, with minimal added benefit. We plan to explore the use of visual information in future work.

CraftDroid and TreaDroid each migrated two unique test scripts, respectively. Also, one test script was successfully migrated uniquely by both of them. Upon inspection, we found that the cause was widget matching differences. In these cases, misleading elements in the GUI screen led to incorrect matches by ITeM, while CraftDroid and TreaDroid identified the correct GUI elements. Figure 4 shows an example of incorrect migrations produced by ITeM. The test intention specified in this case is "*confirm to save a list item in the to-do list app*". In the source test script, pressing the "OKAY" button (id: "*bt_new_task_ok*", text: "*Okay*") navigates the source app from S1 to S2. In the target app, CraftDroid and TreaDroid correctly matched the original button to the "*fab_add*" button, navigating the target app from S3 to S5 where a new list item is added. In contrast,

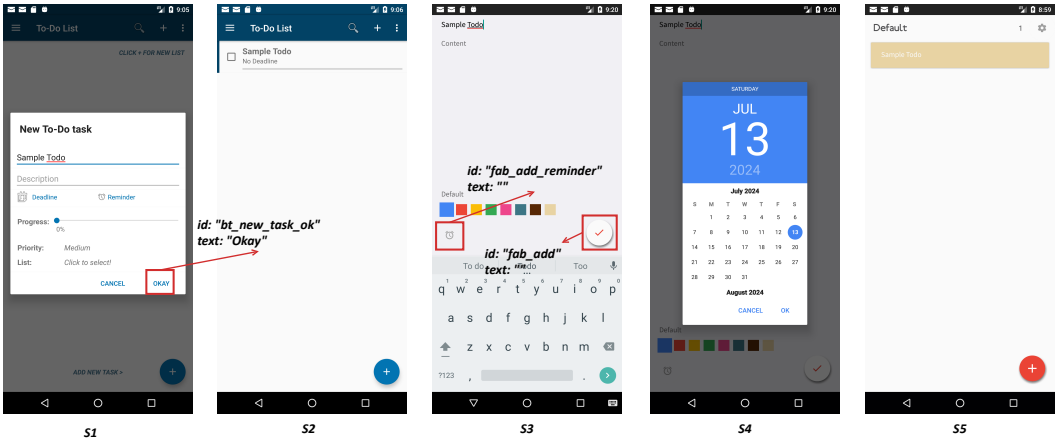


Fig. 4. An example of incorrect migrations produced by ITeM

ITeM mistakenly interacted with the "*fab_add_reminder*" button, navigating to S4 (the reminder selection screen). This error likely stemmed from the limited information available to the LLM, as the attributes alone provided no clear distinction between "*fab_add*" and "*fab_add_reminder*." Consequently, the LLM incorrectly assumed "*fab_add_reminder*" related to adding list items rather than date reminders. We manually replayed this case, providing images of both buttons to the LLM, which then correctly identified the match, supporting our assumption. Future improvements may involve incorporating more widget information.

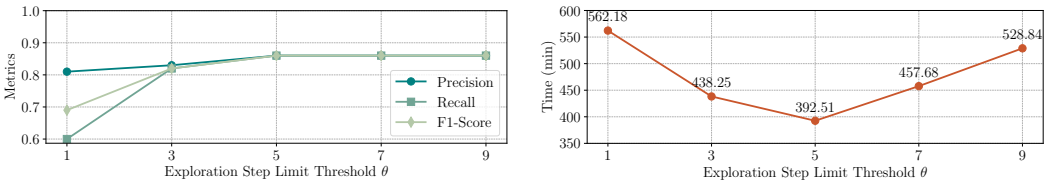
Regarding the migration capabilities of the remaining approaches, ATM performed the worst on our dataset, yielding the fewest successful migrations and failing to produce unique migrations. Additionally, most of its migrations were already covered by other approaches. In contrast, CraftDroid and TreaDroid successfully migrated 72 and 79 test scripts, respectively. However, their abilities did not overlap completely, with TreaDroid producing an additional 17 migrations and CraftDroid producing 11. Notably, these 28 migrations were fully covered by ITeM, which successfully migrated 150 test scripts—surpassing the combined total of CraftDroid and TreaDroid. This demonstrates ITeM's superior effectiveness.

The exploration aspect of our approach may occasionally introduce irrelevant GUI events during migrations. Upon inspecting the successful migrations, we found that 20% (30/150) of migration tasks included exploration events. Of these, 22 tasks were fully correct, with exploration events aligning with those in the ground truth. In the remaining eight tasks, 22 irrelevant GUI events were generated. These events, unrelated to the original test intentions, served as exploratory steps that navigated the app to correct GUI states, facilitating the migration process. These irrelevant events had minimal impact on the overall success of the migration, and removing them manually is straightforward. Consequently, we argue that these irrelevant GUI events will not influence the overall success of the migration.

4.3.3 RQ3-Efficiency. As shown in Table 5, ITeM completes the migration of all test scripts in 392.51 minutes, which is faster than all other approaches except AdbGPT. However, since AdbGPT only migrates GUI events and does not handle GUI assertions, its overall time cost cannot be directly compared. The efficiency difference between ITeM and other comparable approaches is attributable to ITeM's novel migration framework. The other migration approaches restart the app each time they try to produce a matching widget and perform an action on it. This restart process aims to backtrack failed migration attempts to improve effectiveness, but it is time-consuming as it often

Table 8. Contributions of different components of ITEM

Approach	Precision	Recall	F1	1-n	n-1	1-1
w/o Intention Extraction	0.62	0.50	0.55	8	8	30
w/o Dynamic Reasoning	0.60	0.62	0.61	0	0	65
ITEM	**0.86**	**0.86**	**0.86**	**30**	**29**	**91**



(a) Effectiveness (b) Efficiency

Fig. 5. The comparison of Precision, Recall, F1-score and time influenced by different values of θ

requires multiple rounds of trials to find a correct matching element. For GUIDER, the primary bottleneck is its reliance on visual information. In contrast, ITEM significantly reduces migration time by avoiding frequent restarts to backtrack app states. Instead, it employs reasoning-based prompting for the LLM to explore the app, similar to how experienced human testers navigate through functionalities, thereby greatly reducing time costs. Moreover, ITEM correctly migrates test actions on the first or second attempt for most cases, eliminating the need for additional trials and further reducing time.

4.3.4 RQ4-Ablation Study. To understand the contribution of different components to the overall effectiveness of ITEM, we designed and compared two variants: one without the *Intention Extraction* component and one without the *Dynamic Reasoning* component. Table 8 shows the effectiveness comparison under different settings. As shown in the table, using natural language format intentions significantly improves ITEM's effectiveness, with increases of 0.24, 0.36, and 0.31 in Precision, Recall, and F1-score, respectively. Moreover, the number of successfully migrated scripts also increases significantly. The reason for the improvement is that without natural language form intentions, the raw structured form of intentions includes too much app-specific information, which misleads the LLM, resulting in more incorrect events (FPs) or an inability to find corresponding actions to fulfill that intention (FNs) on the target app. In contrast, natural forms of intention provide fine-grained and clear instructions for the LLM to understand and fulfill test intentions, thus producing superior effectiveness.

Regarding *Dynamic Reasoning*, we observed that without this mechanism, overall Precision, Recall, and F1-score decreased by 0.26, 0.24, and 0.25, respectively, indicating that *Dynamic Reasoning* enhances effectiveness. We also manually checked the migrated scripts produced by this variation. As shown in Table 8, without *Dynamic Reasoning*, no 1-n or n-1 migration can be successfully realized. This is because the *Dynamic Reasoning* enables our approach to explore the app for specific functionality, thus having chances to complete 1-n migration. The exploration step limit allows our approach to skip unfulfillable intentions, thereby facilitating n-1 migrations. Additionally, we observed that not only do the 1-n and n-1 migrations decrease, but the 1-1 migrations decrease as well. This is because the reasoning process enhances the LLM's reasoning capability, thereby increasing overall effectiveness.

Table 9. Comparison of different LLMs on the effectiveness of ITEM

LLM	Precision	Recall	F1-score
GPT-4-turbo	0.86	0.86	0.86
GLM-4	0.80	0.78	0.79
Llama-3.1-70B	0.78	0.79	0.78

Figure 5 shows the metrics of effectiveness and efficiency under the different settings of threshold θ in Algorithm 1. As depicted in the figure, when θ increases, the Precision, Recall, and F1-score also improve until θ reaches 5. This is because a larger θ allows the LLM to perform more exploration actions, enabling it to handle more complex cases. These three metrics stabilize when θ exceeds 5 because the test intentions requiring extensive app exploration are already fulfilled. Our observations indicate that, on average, no more than 5 steps of exploration are needed, so further increasing θ does not reduce FPs or FNs as their root causes cannot be resolved by additional exploration, which is discussed previously.

An interesting trend in time efficiency is observed in Figure 5b. Initially, as θ increases up to 5, the time cost decreases. When θ exceeds 5, the time cost starts to increase. The additional time cost before θ reaches 5 is due to exploration limits preventing some test intentions from being fulfilled correctly, leading to incorrect skips. These more skips lead to more recovery actions, which involve executing the already generated GUI events (lines 21-23 in Algorithm 1). Moreover, these skips impact subsequent test intentions, causing them to require additional actions to be fulfilled or to be incorrectly skipped as well. This cumulative effect leads to more explorations and, consequently, increased migration time. This effect also explains the low scores in Recall before θ reaches 5. As the θ increases from 1 to 5, ITEM correctly fulfills more test intentions that were previously incorrectly skipped due to a lower exploration limit, thereby reducing recovery action and enhancing efficiency. When θ exceeds 5, the exploration process generally remains the same for all values of θ; the only difference in time arises from skipped test intentions that need exploration up to θ. Therefore, as θ increases, the time cost also increases.

Based on the observed differences in effectiveness and efficiency, we set θ to 5 by default. This value optimizes the balance between achieving high Precision, Recall, and F1-score while maintaining reasonable migration times.

4.3.5 RQ5-Generality. Table 9 shows the comparison of the effectiveness of ITEM using different LLMs. As shown in the table, GPT-4-turbo is still the most powerful model in this task, outperforming GLM-4 and Llama3.1-70B by 7% and 8% in F1-score, respectively. Despite differences in model architecture and performance, ITEM still achieved acceptable results with the other two models. Additionally, in our experiments, all LLMs used were universal and had no fine-tuning. Future work could involve fine-tuning LLMs on testing or mobile applications to further improve performance.

5 Threats to Validity

In this section, we discuss potential threats to the validity of our experimental conclusions.

Construct Validity. The primary threat to construct validity comes from the evaluation metrics used. To address this, we employed widely adopted metrics—Precision, Recall, and F1-score to evaluate the effectiveness of the approaches. The calculation method for these metrics is consistent across the comparison approaches, helping to alleviate this threat.

Internal Validity. A potential threat to internal validity is the possibility of defects in our implementation of ITEM. To mitigate this threat, we reviewed and tested our implementation to reduce the risk of incorrectness. Additionally, the inherent randomness of LLMs poses a threat to

internal validity. To address this issue, we carefully designed prompts using different techniques both for the reasoning and output formatting part, enhancing the robustness and consistency of the output of the LLMs. The randomness of LLM generation may also produce different results for the same prompt. To mitigate this, we ran the LLM-powered approaches (i.e., ITeM, AdbGPT, and the ablation variations of ITeM) three times. The metrics were acquired from the aggregation of the three runs.

External Validity. A major threat to external validity is that the apps used in our experiments may not represent the broad spectrum of Android apps. To mitigate this, we first collected apps from two widely used test migration datasets—CraftDroid and the FrUITer dataset. To reduce bias and ensure a more comprehensive evaluation, we also incorporated a new category of newly released apps, which likely represents previously unseen data for the LLM-powered approaches. Furthermore, we plan to conduct larger-scale experiments in the future to evaluate ITeM more thoroughly.

6 Related Work

6.1 GUI Test Migration

In recent years, GUI test migration has attracted significant attention. Many approaches address this challenge by modeling it as a widget-matching problem.

AppFlow [10] is a semi-automated machine-learning-based method that maps app-specific events to predefined manual events through a multi-label classifier. AppTestMigrator [3] determines matching relationships of widgets by calculating both the edit distance and Word2Vec similarity of them. Additionally, it utilized Gator [28], a static analysis tool that captures Window Transition Graphs (WTGs) of Android apps, to expand the search space for widget matching. CraftDroid [16] also employs the Word2Vec model to compute the similarity of widget textual information. It extracts a UITG from the source code to represent interaction flows and screen transitions triggered by specific widgets. This UITG is dynamically updated during the migration process. TRASM [17] optimizes widget-matching results produced by Word2Vec through conducting a rematching process to identify widgets with higher similarity scores, thus producing more reliable matches. TEMdroid [33] is a learning-based widget-matching approach that fine-tunes BERT [7]. This approach incorporates contextual information of widgets and employs a two-stage training strategy to handle hard-negative samples, thus achieving promising results. TreaDroid [18] combines GUI event deduplication with an enhanced adaptive semantic matching strategy to enhance the effectiveness of widget-matching test migration.

Unlike these approaches, our approach ITeM takes a unique pathway. We conclude that the core of the test migration task is to fulfill the same test intentions across different apps. Thus, ITeM extracts test intentions from the source app's test scripts and fulfills these intentions on the target app, completing the test case migration.

There are also alternative approaches to widget matching for GUI test migration. MigrationPro [34] is a synthesis-based approach that aggregates multiple migrated test cases to produce more reliable outcomes. It has proven effective in enhancing several existing migration techniques. In the future, we aim to explore the possibility of incorporating it into our framework. MACdroid [32] is a parallel work to ITeM that was created concurrently. Both approaches share the core idea of leveraging LLMs for test migration tasks, but the implementation details differ significantly. For instance, MACdroid first aggregates multiple test cases from different apps for the same functionality, then extracts general logic from those test cases with the help of an LLM. It requires multiple well-structured input test cases and generates logic that adheres to the same structure. In contrast, our approach does not rely on multiple test cases and can generate general test logic from a single test

case for a specific app. Additionally, ITEM do not require well-structured test logic; the input is simply the execution trace of the source test, and the output is in natural language, not constrained by a specific structure. While the test intention extraction task in ITEM may be more complex than in MACdroid, it still yields effective outcomes. We would compare the two approaches more thoroughly in future work.

Empirical studies on GUI test migration also exist. Zhao et al. [36] proposed a framework for evaluating GUI test reuse approaches and standardizing the metrics. Mariani et al. [22] evaluated the effectiveness of AppTestMigrator and CraftDroid, revealing the impact of incorporating different widget attributes in the widget-matching process.

6.2 LLM for Mobile Applications

Recent years have witnessed the success of LLMs in NLP tasks, encouraging researchers to explore their potential in addressing issues within the domain of mobile apps. InputBlaster [21] leverages prompt engineering and mutation rules to automatically generate unusual text inputs for testing mobile apps, aiming to provoke potential app crashes. HintDroid [20] automatically generates hint text using LLMs when this information is omitted by developers. GPTDroid [19] is an automated GUI testing approach that iteratively provides GUI screens to the LLM and requests further actions for execution. It serves as a general testing tool without a specific focus on testing, aiming for thorough app testing with high coverage. Compared with it, our tool is specifically designed for test migration, which involves fulfilling particular test intentions and therefore requires more carefully crafted prompts and feedback mechanisms. ADBGPT [9] is an approach that automatically reproduces the bug reports of mobile apps by first extracting steps in a structured format from the bug reports and then prompting the LLM to execute them. While both our work and ADBGPT prompt the LLM to achieve specific testing goals, there are significant differences. First, ADBGPT [9] uses a structured format for the steps, which has been found to be ineffective than using natural language in test migration tasks (Section 4.3.4). Additionally, reproducing a bug report for a specific app version involves less complex exploration, as it requires following the steps outlined in the bug report without deviations. Consequently, ADBGPT employs a simple but effective backtracking strategy for exploration that reverts to the previous step for a second attempt when the current attempt fails. However, this approach is less effective in test migration, where some intentions require multiple actions to be fulfilled. Reverting only one step is inefficient and insufficient for these complex scenarios, making it less suitable for achieving correct outcomes in test migration. In contrast, our design for ITEM enables the LLM to generate multiple exploration actions until an intention action is produced, effectively addressing the aforementioned problem.

7 Conclusion

In this paper, we present ITEM, a novel framework that extracts the test intentions of the source test script and then fulfills them on the target app for GUI test migration. Experimental evaluation on 35 apps across seven categories demonstrates ITEM's superior effectiveness and efficiency over SOTA approaches.

8 Data Availability

The dataset used in our experiment and the source code of our artifact, are publicly available online[1].

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This research is supported by the National Natural Science Foundation of China under Grant No. 62372227.

References

[1] 2024. Appium. http://appium.io/

[2] 2024. Google Play Statistics and Trends 2024. https://42matters.com/google-play-statistics-and-trends

[3] Farnaz Behrang and Alessandro Orso. 2019. Test Migration Between Mobile Apps with Similar Functionality. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 54–65. doi:10.1109/ASE.2019.00016

[4] Shaoheng Cao, Minxue Pan, Yu Pei, Wenhua Yang, Tian Zhang, Linzhang Wang, and Xuandong Li. 2024. Comprehensive Semantic Repair of Obsolete GUI Test Scripts for Mobile Applications. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 90:1–90:13. doi:10.1145/3597503.3639108

[5] Jiuhai Chen, Lichang Chen, Heng Huang, and Tianyi Zhou. 2023. When do you need Chain-of-Thought Prompting for ChatGPT? *CoRR* abs/2304.03262 (2023). doi:10.48550/ARXIV.2304.03262 arXiv:2304.03262

[6] Xiang Chen, Ningyu Zhang, Xin Xie, Shumin Deng, Yunzhi Yao, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2022. KnowPrompt: Knowledge-aware Prompt-tuning with Synergistic Optimization for Relation Extraction. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini (Eds.). ACM, 2778–2788. doi:10.1145/3485447.3511998

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. doi:10.18653/V1/N19-1423

[8] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024). doi:10.48550/ARXIV.2407.21783 arXiv:2407.21783

[9] Sidong Feng and Chunyang Chen. 2024. Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 67:1–67:13. doi:10.1145/3597503.3608137

[10] Gang Hu, Linjie Zhu, and Junfeng Yang. 2018. AppFlow: using machine learning to synthesize robust, reusable UI tests. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 269–282. doi:10.1145/3236024.3236055

[11] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. 2013. Real Challenges in Mobile App Development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, October 10-11, 2013*. IEEE Computer Society, 15–24. doi:10.1109/ESEM.2013.9

[12] Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tennenholtz. 2022. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *CoRR* abs/2205.00445 (2022). doi:10.48550/ARXIV.2205.00445 arXiv:2205.00445

[13] Pavneet Singh Kochhar, Ferdian Thung, Nachiappan Nagappan, Thomas Zimmermann, and David Lo. 2015. Understanding the Test Automation Culture of App Developers. In *8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015*. IEEE Computer Society, 1–10. doi:10.1109/ICST.2015.7102609

[14] Chun Li, Yifan Xiong, Zhong Li, Wenhua Yang, and Minxue Pan. 2023. Mobile Test Script Generation from Natural Language Descriptions. In *23rd IEEE International Conference on Software Quality, Reliability, and Security, QRS 2023,*

Chiang Mai, Thailand, October 22-26, 2023. IEEE, 348–359. doi:10.1109/QRS60937.2023.00042

[15] Haonan Li, Yu Hao, Yizhuo Zhai, and Zhiyun Qian. 2024. Enhancing Static Analysis for Practical Bug Detection: An LLM-Integrated Approach. *Proc. ACM Program. Lang.* 8, OOPSLA1, Article 111 (apr 2024), 26 pages. doi:10.1145/3649828

[16] Jun-Wei Lin, Reyhaneh Jabbarvand, and Sam Malek. 2019. Test Transfer Across Mobile Apps Through Semantic Mapping. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 42–53. doi:10.1109/ASE.2019.00015

[17] Shuqi Liu, Yu Zhou, Tingting Han, and Taolue Chen. 2022. Test Reuse based on Adaptive Semantic Matching across Android Mobile Applications. In *22nd IEEE International Conference on Software Quality, Reliability and Security, QRS 2022, Guangzhou, China, December 5-9, 2022*. IEEE, 703–709. doi:10.1109/QRS57517.2022.00076

[18] Shuqi Liu, Yu Zhou, Longbing Ji, Tingting Han, and Taolue Chen. 2024. Enhancing test reuse with GUI events deduplication and adaptive semantic matching. *Sci. Comput. Program.* 232 (2024), 103052. doi:10.1016/J.SCICO.2023.103052

[19] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2024. Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 100:1–100:13. doi:10.1145/3597503.3639180

[20] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Yuekai Huang, Jun Hu, and Qing Wang. 2024. Unblind Text Inputs: Predicting Hint-text of Text Input in Mobile Apps via LLM. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski (Eds.). ACM, 51:1–51:20. doi:10.1145/3613904.3642939

[21] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Zhilin Tian, Yuekai Huang, Jun Hu, and Qing Wang. 2024. Testing the Limits: Unusual Text Inputs Generation for Mobile App Crash Detection with Large Language Model. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 137:1–137:12. doi:10.1145/3597503.3639118

[22] Leonardo Mariani, Ali Mohebbi, Mauro Pezzè, and Valerio Terragni. 2021. Semantic matching of GUI events for test reuse: are we there yet?. In *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, Cristian Cadar and Xiangyu Zhang (Eds.). ACM, 177–190. doi:10.1145/3460319.3464827

[23] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). doi:10.48550/ARXIV.2303.08774 arXiv:2303.08774

[24] Yisheng Song, Ting Wang, Puyu Cai, Subrota K. Mondal, and Jyoti Prakash Sahoo. 2023. A Comprehensive Survey of Few-shot Learning: Evolution, Applications, Challenges, and Opportunities. *ACM Comput. Surv.* 55, 13s (2023), 271:1–271:40. doi:10.1145/3582688

[25] Mario Linares Vásquez, Carlos Bernal-Cárdenas, Kevin Moran, and Denys Poshyvanyk. 2017. How do Developers Test Android Applications?. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*. IEEE Computer Society, 613–622. doi:10.1109/ICSME.2017.47

[26] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.).

[27] Tongtong Xu, Minxue Pan, Yu Pei, Guiyin Li, Xia Zeng, Tian Zhang, Yuetang Deng, and Xuandong Li. 2021. GUIDER: GUI structure and vision co-guided test script repair for Android apps. In *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, Cristian Cadar and Xiangyu Zhang (Eds.). ACM, 191–203. doi:10.1145/3460319.3464830

[28] Shengqian Yang, Hailong Zhang, Haowei Wu, Yan Wang, Dacong Yan, and Atanas Rountev. 2015. Static Window Transition Graphs for Android (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 658–668. doi:10.1109/ASE.2015.76

[29] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.).

[30] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=WE_vluYUL-X

[31] Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. *CoRR* abs/2406.12793 (2024). doi:10.48550/ARXIV.2406.12793 arXiv:2406.12793

[32] Yakun Zhang, Chen Liu, Xiaofei Xie, Yun Lin, Jin Song Dong, Dan Hao, and Lu Zhang. 2024. LLM-based Abstraction and Concretization for GUI Test Migration. *CoRR* abs/2409.05028 (2024). doi:10.48550/ARXIV.2409.05028 arXiv:2409.05028

[33] Yakun Zhang, Wenjie Zhang, Dezhi Ran, Qihao Zhu, Chengfeng Dou, Dan Hao, Tao Xie, and Lu Zhang. 2024. Learning-based Widget Matching for Migrating GUI Test Cases. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 69:1–69:13. doi:10.1145/3597503.3623322

[34] Yakun Zhang, Qihao Zhu, Jiwei Yan, Chen Liu, Wenjie Zhang, Yifan Zhao, Dan Hao, and Lu Zhang. 2024. Synthesis-Based Enhancement for GUI Test Case Migration. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, Maria Christakis and Michael Pradel (Eds.). ACM, 869–881. doi:10.1145/3650212.3680327

[35] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. *CoRR* abs/2303.18223 (2023). doi:10.48550/ARXIV.2303.18223 arXiv:2303.18223

[36] Yixue Zhao, Justin Chen, Adriana Sejfia, Marcelo Schmitt Laser, Jie Zhang, Federica Sarro, Mark Harman, and Nenad Medvidovic. 2020. FrUITeR: a framework for evaluating UI test reuse. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1190–1201. doi:10.1145/3368089.3409708

Received 2024-10-31; accepted 2025-03-31