Context-Aware GUI Testing for Mobile Applications

Presenters: Buochari Youness, Bergamini Luca, Bollo Matteo

Goals

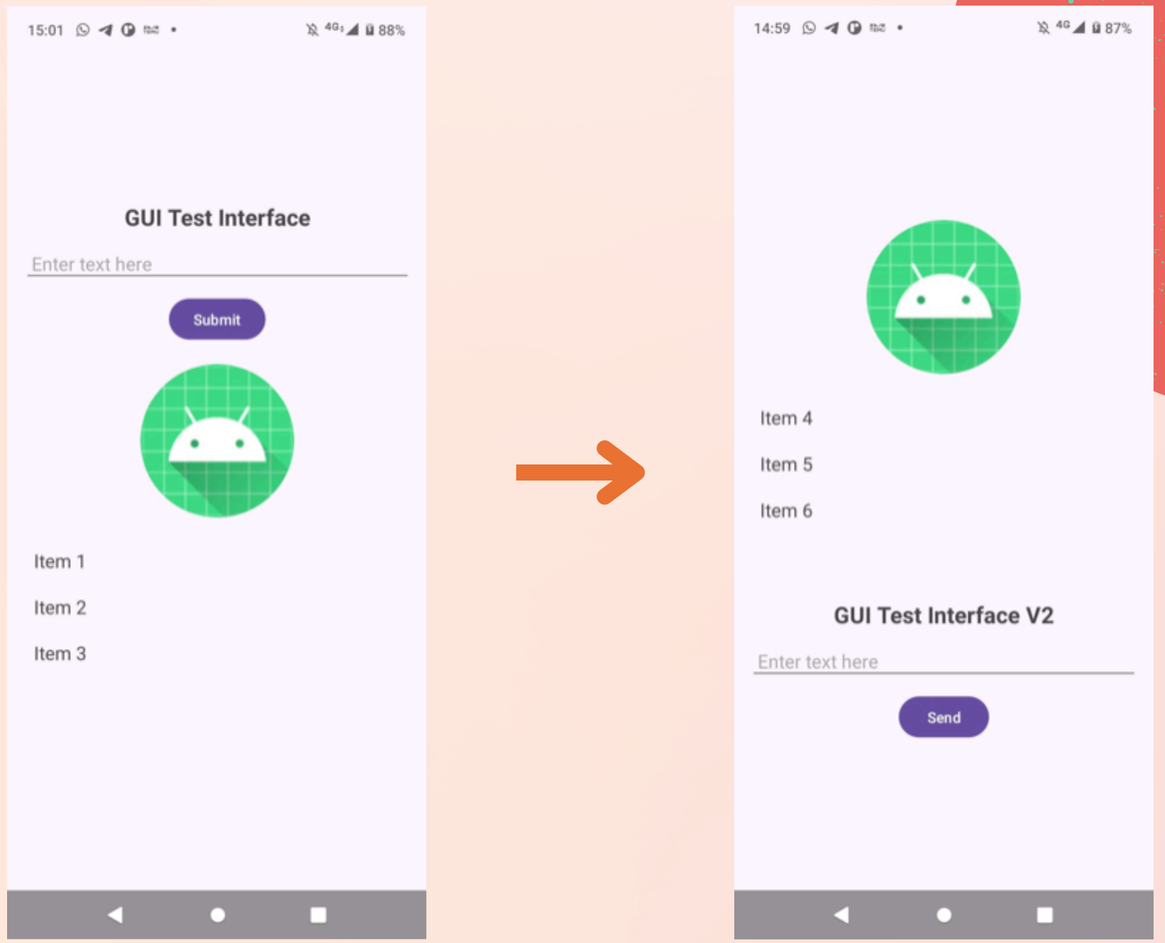
Mobile Applications GUI have frequent changes, we aim to:

- Asses LLMs capability in recognizing changes
- Asses LLMs capability in generating test for the new GUI

GUIs

To fully test LLM capability, we generated our GUI application and tried to change as much as possible:

- Widget’s content
- Widget’s order
- Widget’s ids



OmniNotes

On the other hand, for a real case scenario, we decided to use OmniNotes, an open-source Android application (a clichè for research purposes). It is composed of different widgets that need to be tested.

Appium

Here comes some complications. We tried several versions, several releases and several repositories, well running the tests is not trivial. We decided to automatize the retrieval of the XML from the APK of the application using the Appium framework.

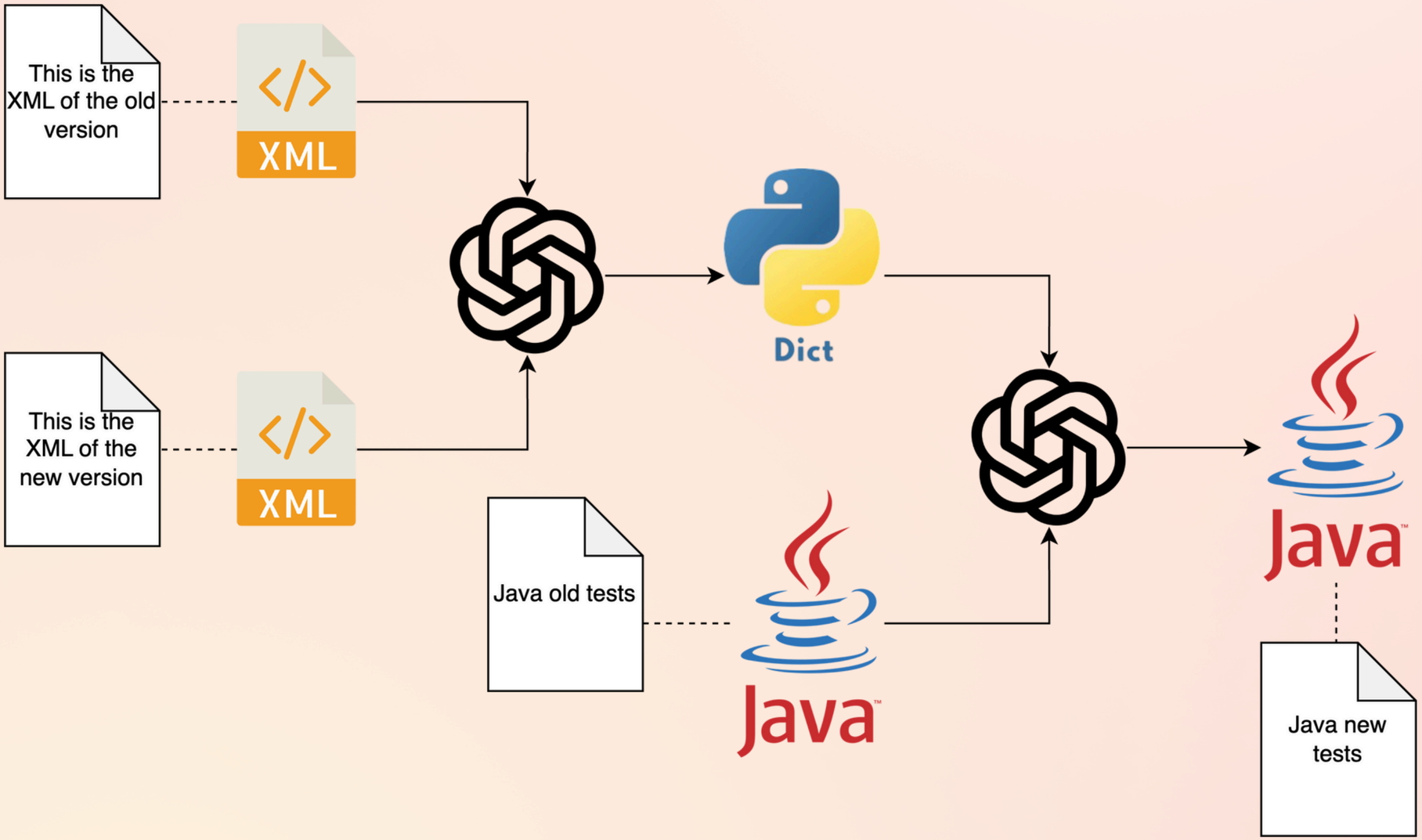
```
try:
    while True:
        # Screenshot XML
        xml_source = driver.page_source
        xml_filename = f"xmlScreen/screen{counter}.xml"  # Salva in xmlScreen
        with open(xml_filename, "w", encoding="utf-8") as f:
            f.write(xml_source)
        print(f"✅ Screenshot XML salvato come {xml_filename}")

        # Screenshot JPG
        screenshot_filename = f"jpgScreen/screen{counter}.jpg"  # Salva in jpgScreen
        driver.get_screenshot_as_file(screenshot_filename)
        print(f"✅ Screenshot JPG salvato come {screenshot_filename}")

        # Incrementa il contatore per il prossimo file
        counter += 1

        # Aspetta 5 secondi prima di acquisire il prossimo screenshot
        time.sleep(5)
```

Architecture



```
prompt = """
You are an expert in analyzing Android UI XML layouts.
Your task is to compare two XML layouts and find the differences between them.

## Comparison Task
Here is the old UI XML layout:
"""+ old_xml_content + """

Here is the updated UI XML layout:
"""+ new_xml_content +"""

Please list all UI changes between these two versions.
Format the response as structured text, including:
- Elements that were **added** (with their IDs and attributes).
- Elements that were **removed**.
- Elements that were **renamed**.
- Any **structural changes** (e.g., button moved inside another layout).
- Any **attribute changes** (e.g., text color changed).
- Any **value changes**, checking the text content, checking the text attribute i
For example if an item, in the old version, have the attribute "text="Item 1" and
to write that as a difference.

Return the response in JSON format.

## Example 1: Simple Button Change
Old XML:
```xml
<Button
    android:id="@+id/submit_button"
    android:layout_width="wrap_content"
    android:text="Send"
    android:background="#FF0000" />
```

New XML:
```xml
<Button
```
```

Prompt 1

The effectiveness of this method relies on accurate few-shots prompts for the LLM to execute

```
prompt = """
You are an expert in Android UI test automation using Espresso.
The Android app's UI has changed, and test cases need to be updated accordingly.

## Actual Test Case Update Task
Here are all the detected UI changes:
"""+ json.dumps(xml_differences, indent=4) +"""

Here is an old test case:
"""+ old_test +"""

Please update the test case to ensure it remains valid with the new UI structure.
Ensure that:
- It still verifies the original functionality.
- It interacts with the correct elements based on the updated XML.
- Any removed elements are handled gracefully.
- Any renamed or moved elements are adjusted accordingly.
- It follows best practices for Android UI test automation.

Return only the updated Java test code without any explanation.

## Example 1: Simple Button Change
XML differences:
```json
{
    "changes": {
        "added": [],
        "removed": [],
        "renamed": [],
        "structural_changes": [],
        "attribute_changes": [
            {
                "id": "@+id/submit_button",
                "changes": [
                    {
                        "attribute": "android:layout_width",
                        "old_value": "wrap_content",
```

Prompt 2

The effectiveness of this method relies on accurate few-shots prompts for the LLM to execute
```

Evaluation

We evaluated the LLMs capabilities of:

- Recognizing GUI differences.
- Generating new tests for the new GUI version.

For test generation, we compare the tests with the ground truth of the actual tests. To have unbiased results towards what we think is correct we checked whether the tests contained the same words.

```
ai_tokens = tokenize(ai_test)
human_tokens = tokenize(human_test)

ai_counter = Counter(ai_tokens)
human_counter = Counter(human_tokens)

# compute intersection: words that appear in both tests
common_tokens = sum((ai_counter & human_counter).values())

precision = common_tokens / sum(ai_counter.values()) if ai_counter else 0
recall = common_tokens / sum(human_counter.values()) if human_counter else 0
```

Results

- Regarding the recognition of the differences, leveraging our custom application, we find 100% of the visual differences in the generated XML file.
- For the test generation, our custom application still obtained maximum results, while OmniNotes (comparing word x word) we achieved:
  - precision: 98.24%
  - recall: 73.39%

Test	Precision One shot	Recall One shot
CategoryLifecycleTest.java	0.9987	0.8588
SearchTagBackArrowTest.java	0.8596	0.4558
NoteLifecycleTest.java	0.9790	0.6619
FabCameraNoteTest.java	1.0000	0.6452
AutoBackupTest.java	1.0000	0.8624
RecurrenceRuleTest.java	0.9956	0.6338
SettingsActivityTest.java	0.9932	0.8420
RemindersLifecycleTest.java	1.0000	0.6254
BaseEspressoTest.java	0.9914	0.7670
MrJingleLifecycleTest.java	0.9699	0.7655
NoteListMenuTest.java	0.8543	0.5244
DrawerMenusEspressoTest.java	0.9124	0.5208
FabLifecycleTest.java	0.9897	0.5884

Results

Since we now LLM are few shot learner we improve our results with higher quality prompts obtaining an overall of :

- precision: 98.32%
- recall: 96.60%

Test	Precision Few Shot	Recall Few Shot
CategoryLifecycleTest.java	0.9986	0.9893
SearchTagBackArrowTest.java	0.8793	1.0000
NoteLifecycleTest.java	0.9958	0.9958
FabCameraNoteTest.java	0.9870	0.9870
AutoBackupTest.java	1.0000	1.0000
RecurrenceRuleTest.java	0.9959	0.9959
SettingsActivityTest.java	1.0000	1.0000
RemindersLifecycleTest.java	0.9948	0.9095
BaseEspressoTest.java	0.9485	0.9984
MrJingleLifecycleTest.java	0.9859	0.8108
NoteListMenuTest.java	0.9318	0.9919
DrawerMenusEspressoTest.java	0.9920	0.9764
FabLifecycleTest.java	0.9602	0.8977

Future works

- Despite the nice results, this method comes with some limitations, relying on the XML of the GUI might, so a textual representation, might not be enough for some peculiar graphical changes, here a VLLM can give us a hand.
- For complex GUI instead on relying on prompt engineering a fine-tuned LLM can be an improvement.

Thank You

Buochari Youness, Bergamini Luca
and Bollo Matteo