Context-Aware GUI Testing for Mobile Applications

Raffaele Martone, Michele Russo, Giuseppe Monteasi

06/07/2025

1 Introduction

In recent years, testing Graphical User Interfaces (GUIs) for mobile applications has faced increasing challenges due to the diversity and complexity of mobile app interactions. Traditional automated testing tools often lack the adaptability to handle context-specific user interactions, dynamic content, and the wide variety of screen sizes and resolutions in mobile devices. In this context, the application of Large Language Models (LLMs) has emerged as a promising approach for enhancing GUI testing by making it more context-aware and adaptable to real-world scenarios.

LLMs, trained on extensive datasets of language and user interactions, offer the potential to understand user intent and app behavior within different contexts. This ability allows LLMs to simulate realistic user interactions and predict potential edge cases that are difficult to capture with rule-based automation alone. For example, LLMs can generate test cases that respond to contextual cues, such as location, app permissions, or user settings, and adapt these tests as app states change dynamically.

The main ideas behind using LLMs for context-aware GUI testing include:

Contextual Interaction Simulation LLMs can simulate user interactions that adapt to specific contexts, such as user profiles, permissions, and app state, offering insights into real-life usage patterns.

Dynamic Test Case Generation By generating test cases that are flexible and responsive to changes in app behavior and layout, LLMs can improve coverage and reveal defects that would typically require extensive manual testing.

The objective of this research is to investigate the potential of LLMs to support context-aware GUI testing in mobile applications, with the goal of enhancing test coverage, improving the accuracy of test results, and reducing the time and resources required for manual testing. By exploring the integration of LLMs into mobile testing frameworks, this study aims to offer insights into

how AI-driven testing can support the development of more resilient and user-friendly mobile applications.

2 Research questions

RQ1: How effectively can LLMs generate working context-aware test cases for mobile GUI testing?

RQ2: What coverage can be obtained by generating test cases with LLMs?

3 Background and Related Work

3.1 Automated Mobile GUI Testing

3.1.1 Rule-based and Model-based Testing

Il testing delle interfacce grafiche mobili si basa tradizionalmente su framework come Espresso, che permettono di automatizzare l'interazione con le applicazioni Android attraverso script predefiniti. Questi strumenti consentono di simulare azioni dell'utente (come click, swipe e inserimento di testo) e di verificare il corretto comportamento dell'applicazione in risposta a tali input. Tuttavia, la creazione manuale di questi script risulta spesso onerosa e poco scalabile, soprattutto in presenza di applicazioni complesse o soggette a frequenti aggiornamenti.

Un ulteriore limite di framework come Espresso è la necessità di adottare un approccio open box: per scrivere test efficaci è indispensabile una conoscenza approfondita del codice sorgente dell'applicazione. Questo requisito implica non solo l'accesso diretto al codice, ma anche la comprensione della sua struttura interna, delle dipendenze e dei flussi logici. In contesti reali, la qualità della documentazione può essere insufficiente e il codice spesso risulta poco leggibile o scarsamente commentato, aggravando ulteriormente la difficoltà di scrivere e mantenere test affidabili.

Per superare queste criticità, sono stati sviluppati approcci rule-based e model-based, che mirano ad automatizzare la generazione dei casi di test sfruttando regole statiche o modelli di navigazione della GUI.

3.1.2 DroidBot Framework

DroidBot è uno strumento open source per la generazione automatica di input di test su applicazioni Android, progettato per essere leggero e facilmente utilizzabile senza la necessità di strumentare né l'applicazione né il sistema operativo. La sua principale caratteristica distintiva è la capacità di guidare l'esplorazione della GUI tramite un modello di transizione di stato costruito dinamicamente durante l'esecuzione dell'app [1] .

A differenza di strumenti come Monkey, che generano sequenze di eventi completamente casuali, DroidBot analizza la gerarchia delle UI e monitora lo

stato del dispositivo e dell'applicazione in tempo reale. Utilizzando queste informazioni, costruisce un grafo orientato in cui ogni nodo rappresenta uno stato della UI e ogni arco un evento di input che ha portato a una transizione di stato. L'esplorazione avviene tipicamente con una strategia depth-first, che permette di andare in profondità alla ricerca di nuovi edge e stati non ancora visitati.

Inoltre, DroidBot registra screenshot, albero della UI, informazioni sui processi e log di sistema per ogni stato, fornendo così una base ricca per l'analisi post-esecuzione.

Grazie a questo approccio, DroidBot è in grado di coprire un maggior numero di stati rispetto a generatori randomici, risultando particolarmente efficace nell'identificare comportamenti sensibili o malevoli che emergono solo in specifiche condizioni di utilizzo[1].

3.1.3 Humanoid Framework

Humanoid è un input generator avanzato per il testing automatico di app Android, progettato per superare i limiti degli approcci randomici e model-based tradizionali. La sua principale innovazione consiste nell'integrare un modello di deep learning, addestrato su centinaia di migliaia di tracce di interazione umana (dataset Rico), per prevedere quali elementi della GUI sono più probabilmente selezionati dagli utenti reali e con quale tipo di azione (touch, swipe, input di testo, ecc.)[2].

Dal punto di vista architetturale, Humanoid si integra direttamente con DroidBot, sostituendo la logica di selezione degli input: mentre DroidBot esplora la GUI costruendo dinamicamente un grafo degli stati (UTG), Humanoid utilizza il suo modello neurale per assegnare una probabilità a ciascuna possibile azione in base al contesto corrente della UI. In particolare, il modello di Humanoid sfrutta una combinazione di reti convoluzionali (per estrarre feature visive dalla struttura della UI) e moduli LSTM (Long Short-Term Memory) residui, che consentono di modellare le sequenze di transizioni tra stati e catturare pattern di interazione ricorrenti [2]

Il modello, addestrato sulle tracce di interazione umana, genera degli input durante il test. L'algoritmo di esplorazione di Humanoid seleziona quindi l'azione con la probabilità più alta tra quelle ancora inesplorate, oppure naviga verso stati con nuove azioni da esplorare, ottimizzando così la copertura delle funzionalità più rilevanti dal punto di vista dell'utente.

Grazie a questa architettura, Humanoid è in grado di produrre sequenze di input più simili a quelle di un utente reale, raggiungendo stati della GUI che spesso sfuggono agli approcci puramente randomici o basati su regole statiche[2].

3.1.4 Current Limitations in Context Awareness

Nonostante i notevoli progressi ottenuti dai framework di testing automatico, presentano ancora limiti significativi nella capacità di adattarsi dinamicamente al contesto dell'applicazione. In particolare, la gestione degli input testuali rappresenta una delle principali criticità: la maggior parte degli strumenti, inclusi

DroidBot e Humanoid, si limita a inserire valori placeholder generici come "Hello World" o stringhe predefinite, senza considerare la semantica richiesta dai diversi campi di input o la varietà di combinazioni possibili. Questo approccio riduce sensibilmente la capacità di esplorare percorsi alternativi nella GUI e di rilevare bug o comportamenti anomali che emergono solo in presenza di valori specifici. La mancanza di consapevolezza del contesto e di generazione semantica degli input testuali evidenzia la necessità di soluzioni più avanzate e flessibili, in grado di adattare dinamicamente i valori inseriti in base alla struttura e alla logica dell'applicazione sotto test. [**wang2024softwaretestinglargelanguage**]

3.2 LLMs for Automated and Context-Aware Testing

3.2.1 Overview of LLM Applications in Software Testing

Negli ultimi anni, l'adozione dei Large Language Models (LLM) ha rivoluzionato il panorama del software testing, offrendo nuove opportunità per automatizzare e potenziare numerose attività di verifica e validazione. Secondo la survey di Wang et al., gli LLM sono stati impiegati con successo in una vasta gamma di task, tra cui la generazione automatica di casi di test (sia a livello di unit che di sistema), la generazione di test oracle, il debug, la riparazione automatica di bug, l'analisi di bug report e la generazione di script di test per GUI e API[5]. In particolare, la generazione di casi di test e la preparazione di input per test di sistema rappresentano due delle aree più promettenti, dove l'intelligenza generativa degli LLM consente di superare i limiti degli approcci tradizionali basati su regole o su euristiche[5].

3.2.2 Prompt Engineering, In-Context Learning, and Advanced LLM Techniques

L'efficacia degli LLM nel software testing dipende fortemente dalle tecniche di interazione adottate. La survey di Wang et al. evidenzia che, accanto alle strategie di pre-training e fine-tuning, la maggior parte degli studi recenti sfrutta il prompt engineering per guidare il comportamento degli LLM verso risultati desiderati[5]. Le strategie più diffuse includono lo zero-shot e il few-shot learning, in cui il modello riceve istruzioni o esempi direttamente nel prompt per generare test case, script di test o input specifici. In particolare, QTypist[3] dimostra come la costruzione automatica di prompt contestuali, arricchiti da informazioni locali e globali estratte dalla GUI, possa aumentare drasticamente la qualità e la pertinenza degli input generati, permettendo di superare le limitazioni degli approcci statici o euristici.

3.2.3 Context Awareness and Adaptive Test Generation

Uno dei principali vantaggi degli LLM rispetto ai metodi tradizionali è la capacità di comprendere e adattarsi dinamicamente al contesto applicativo, generando input e scenari di test che riflettono l'intento dell'utente e la logica specifica

dell'applicazione. Studi come QTypist[3] e GPTDroid[5][4] mostrano che, formulando la generazione di input come un task di question answering o fill-in-the-blank, è possibile sfruttare la conoscenza semantica e la flessibilità degli LLM per produrre test case adattivi e realistici. Questo approccio consente di superare il tipico ostacolo dei test automatici sulle GUI, ovvero l'impossibilità di procedere oltre schermate che richiedono input specifici. Inoltre, la capacità degli LLM di simulare scenari realistici e di adattare la generazione di test case in base ai feedback ricevuti dall'applicazione apre nuove prospettive per il testing automatico, sia in ambito mobile che in altri domini software.

3.3 LLM-Enhanced GUI Testing: State of the Art

3.3.1 GPTDroid: Functionality-Aware Decisions

GPTDroid [4] propone un approccio innovativo per il testing automatico delle GUI mobili formulando il problema come un task di Question & Answering (Q&A) in cui un LLM interagisce direttamente con l'applicazione. Il sistema estrae informazioni semantiche dettagliate dalla GUI e dall'app, che vengono convertite in prompt linguistici per guidare il modello nella generazione di script di test eseguibili. Una caratteristica chiave di GPTDroid è la memoria funzionalità-consapevole, che permette al modello di mantenere e ragionare sul progresso del testing a livello funzionale, superando così i limiti delle strategie basate solo su dati recenti o a basso livello.

Questa memoria funzionalità-consapevole registra le funzioni già testate, le attività visitate e le operazioni recenti, fornendo al LLM una visione globale e a lungo termine del processo di testing. Grazie a ciò, GPTDroid è in grado di generare sequenze di azioni più umane e mirate, priorizzando le funzionalità chiave dell'app e migliorando significativamente la copertura delle attività (fino al 75%, con un incremento del 32% rispetto ai migliori baseline) e la rilevazione di bug (31% in più) su un ampio set di app reali. Inoltre, GPTDroid supporta azioni composte e input testuali validi, riuscendo a simulare interazioni realistiche e complesse [4].

3.3.2 DroidFiller

DroidFiller [6] affronta una delle principali sfide nel testing automatico delle GUI mobili: la generazione di input testuali semanticamente rilevanti e contestualizzati. Integrando un LLM con una sofisticata tecnica di recupero dinamico del contesto basata su "function calling", DroidFiller permette di generare testi specifici per ogni campo di input, adattandosi al dominio dell'applicazione e al profilo utente simulato. Questo approccio supera i limiti degli input casuali o predefiniti, consentendo di inserire valori come codici coupon o informazioni personali in modo coerente con la logica dell'app.

Grazie a un prompt strutturato che include informazioni sul campo di input, lo stato della GUI e un template di ragionamento ispirato al "Chain-of-Thought", DroidFiller guida il LLM a produrre input coerenti e validi. Inoltre,

la possibilità di invocare funzioni esterne per recuperare dati specifici migliora ulteriormente la qualità degli input generati. [6].

4 Methodology

4.1 Experimental Design

4.1.1 Research Questions Mapping

Il nostro studio si focalizza su due research questions principali che guidano l'intero framework sperimentale. La prima research question (RQ1) "How effectively can LLMs generate working context-aware test cases for mobile GUI testing?" viene valutata attraverso metriche di effectiveness che distinguono tra test cases funzionanti e non funzionanti, analizzando la capacità degli LLM di generare input testuali semanticamente corretti e azioni di testing valide. Per rispondere a questa domanda, misuriamo il tasso di successo dei test generati, la qualità degli input testuali prodotti e la capacità di superare schermate che richiedono input specifici.

La seconda research question (RQ2) "What coverage can be obtained by generating test cases with LLMs?" si concentra su metriche di coverage, includendo il numero di pagine GUI raggiunte e il numero medio di interazioni per pagina. L'obiettivo è quantificare l'incremento di copertura ottenibile attraverso l'integrazione di LLM rispetto agli approcci baseline, valutando sia la breadth (varietà di stati esplorati) che la depth (profondità di esplorazione) del testing automatico.

4.1.2 Hypothesis Formulation

Le nostre ipotesi principali si articolano su due livelli di analisi. La prima ipotesi sostiene che gli approcci LLM-enhanced superino sistematicamente i metodi baseline in termini di effectiveness e coverage, grazie alla capacità di generare input testuali contestualmente appropriati e di prendere decisioni di navigazione più informate. Questa ipotesi si basa sull'assunzione che la conoscenza semantica degli LLM possa compensare le limitazioni degli approcci puramente euristici o casuali.

La seconda ipotesi riguarda l'analisi comparativa tra text generation e decision-making: ipotizziamo che l'integrazione degli LLM per la generazione di testo abbia un impatto significativo sulla coverage, mentre l'utilizzo degli LLM per il decision-making migliori principalmente l'effectiveness attraverso una navigazione più strategica della GUI. Questa distinzione è cruciale per comprendere quali aspetti del testing automatico beneficiano maggiormente dall'integrazione di intelligenza artificiale generativa.

4.2 Subject Applications

Omni-Notes Omni-Notes[1] è un'applicazione open-source per la gestione di note e attività, progettata per offrire un'interfaccia semplice ma potente. Supporta la creazione, modifica e organizzazione di note testuali, checklist, promemoria e allegati multimediali. L'applicazione include funzionalità avanzate come la categorizzazione tramite tag, la ricerca rapida, il supporto a temi Material Design e la possibilità di bloccare note sensibili tramite password. La ricchezza di widget interattivi e la presenza di numerosi flussi di navigazione rendono Omni-Notes un banco di prova ideale per valutare la copertura e la qualità degli input generati dai framework di testing automatico.

PassAndroid PassAndroid[2] è un'applicazione open-source specializzata nella gestione di pass digitali, come biglietti, carte fedeltà, coupon e carte d'imbarco in formato Passbook (*.pkpass) ed esPass (*.esPass). L'app permette di importare, visualizzare e organizzare diversi tipi di credenziali elettroniche, offrendo funzionalità di scansione barcode (QR, PDF417, AZTEC, Code 39, Code 128), accesso completamente offline e un'interfaccia focalizzata sulla semplicità d'uso. PassAndroid si distingue per la gestione di dati dinamici e per la varietà di interazioni tra viste di dettaglio, filtri e funzioni di condivisione, rappresentando così un contesto sfidante per la generazione automatica di test contestuali.

Thunderbird Android (TFA) Thunderbird per Android[3] è un client e-mail open-source, focalizzato su privacy e gestione multi-account. Basato su K-9 Mail, offre funzionalità avanzate tra cui inbox unificata, supporto a notifiche push, filtri, gestione di cartelle e allegati, e una profonda personalizzazione delle impostazioni. L'applicazione presenta scenari di utilizzo complessi, sia pre che post login, con workflow che coinvolgono autenticazione, navigazione tra account, gestione di messaggi e operazioni asincrone di sincronizzazione.

4.3 Testing Framework Setup

4.3.1 Environment Configuration

Il nostro approccio metodologico ha seguito un percorso di sviluppo incrementale, iniziando dall'analisi di framework tradizionali per poi evolversi verso soluzioni LLM-enhanced. La prima fase ha coinvolto lo studio approfondito di Espresso, al fine di comprendere i meccanismi sottostanti alla scrittura manuale di test case e identificare le limitazioni degli approcci open-box. Questa fase preliminare è stata fondamentale per capire effettivamente quanto i successivi approcci cambiassero la vita dei tester.

Successivamente, è stato configurato un ambiente di testing basato su DroidBot come baseline, sfruttando la sua capacità di esplorazione automatica della

[1] `https://github.com/federicoiosue/Omni-Notes`
[2] `https://github.com/ligi/PassAndroid`
[3] `https://github.com/thunderbird/thunderbird-android`

GUI e di generazione di log dettagliati. L'architettura di integrazione con LLAMA è stata progettata per supportare sia deployment tramite API che esecuzione locale, garantendo flessibilità nella gestione delle risorse computazionali e nella configurazione degli esperimenti. L'ambiente supporta inoltre la raccolta sistematica di metriche di performance, log di esecuzione e screenshot per ogni configurazione testata.

4.3.2 Testing Configurations

- **Baseline Approaches:**
  - DroidBot base: configurazione standard per l'esplorazione automatica della GUI
  - Humanoid base: approccio basato su deep learning per la simulazione di interazioni umane
- **Replay-Enhanced Approaches:**
  - DroidBot + Replay output: approccio offline che utilizza LLM per post-processare i file JSON di output, sostituendo input testuali generici con contenuti semanticamente appropriati e poi rilanciare questi test con quest'ultimi.
  - Humanoid + Replay output: combinazione della strategia di esplorazione di Humanoid con post-processing LLM-based degli input testuali
- **LLM-Enhanced Approaches:**
  - DroidBot + LLAMA (text generation): integrazione online di LLAMA nel codice di DroidBot per la generazione dinamica di input testuali durante l'esplorazione
  - DroidBot + LLAMA + Choice (decision making): estensione del precedente con utilizzo di LLAMA anche per la selezione strategica delle prossime azioni da eseguire
  - Humanoid + LLAMA: combinazione della strategia di esplorazione di Humanoid con generazione di input testuali tramite LLAMA

4.4 LLM Integration Strategy

4.4.1 LLM Input Generation

La strategia di integrazione degli LLM si è focalizzata su due approcci complementari: offline e online. Nell'approccio offline (Replay output), i test case generati da DroidBot vengono post-processati analizzando i file JSON di output per identificare eventi di tipo "set$_t ext''contenentiinputgenericicome``HelloWorld''.Questiinputvengonosostitu$

L'approccio online rappresenta un'evoluzione significativa, integrando LLAMA direttamente nel codice di DroidBot per la generazione dinamica di input testuali durante l'esplorazione. Questa integrazione utilizza tecniche di prompt

engineering e few-shot learning per guidare il modello nella generazione di contenuti appropriati al contesto corrente. Il prompt include informazioni estratte in tempo reale dalla GUI, come il tipo di campo di input, label associati, e lo stato dell'applicazione, permettendo una generazione di testo più precisa e contestualmente rilevante rispetto all'approccio offline.

La configurazione più avanzata estende l'utilizzo di LLAMA oltre la generazione di testo, impiegando il modello anche per il decision-making strategico nella selezione delle prossime azioni da eseguire. Questo approccio dual-purpose sfrutta la capacità di ragionamento degli LLM per bilanciare l'esplorazione di nuovi stati GUI con l'approfondimento di funzionalità specifiche, potenzialmente migliorando sia la coverage che l'effectiveness del testing automatico.

5 Results

5.1 Effectiveness in Generating Working Test Cases (RQ1)

5.1.1 Success Rates per Configuration

5.1.2 Working vs. Non-Working Test Case Analysis

5.1.3 Comparison with Baseline Approaches

5.1.4 Quality Assessment of LLM-Generated Tests

5.2 Coverage of Generated Test Cases (RQ2)

5.2.1 GUI Page Coverage Metrics

5.2.2 Average Interactions per Page Analysis

5.2.3 Coverage Comparison Across All Configurations

5.2.4 Manual Inspection Insights

5.3 Performance and Resource Analysis

5.3.1 Execution Time Comparison

5.3.2 Resource Utilization Patterns

5.3.3 Scalability Considerations

5.4 Qualitative Analysis

5.4.1 Types of Errors Encountered

5.4.2 LLM Behavior Patterns

5.4.3 Context Utilization Effectiveness

6 Discussion

Provide in this section a reasoning about the results of your experimentation. Make sure to answer the following questions:

- What are the limitations of the approach you used?
- What further investigations or modifications to the methodology can be performed?
- Is the problem solved? Is it an advantage to perform this problem by using an LLM-based approach instead of performing it manually?

References

[1] Yuanchun Li et al. "DroidBot: a lightweight UI-Guided test input generator for android". In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 2017, pp. 23–26. DOI: 10.1109/ICSE-C.2017.8.

[2] Yuanchun Li et al. "Humanoid: A deep learning-based approach to automated black-box android app testing". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2019, pp. 1070–1073.

[3] Zhe Liu et al. *Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing*. 2022. arXiv: 2212.04732 [cs.SE]. URL: https://arxiv.org/abs/2212.04732.

[4] Zhe Liu et al. *Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions*. 2023. arXiv: 2310.15780 [cs.SE]. URL: https://arxiv.org/abs/2310.15780.

[5] Junjie Wang et al. *Software Testing with Large Language Models: Survey, Landscape, and Vision*. 2024. arXiv: 2307.07221 [cs.SE]. URL: https://arxiv.org/abs/2307.07221.

[6] Juyeon Yoon et al. " Integrating LLM-Based Text Generation with Dynamic Context Retrieval for GUI Testing ". In: *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2025, pp. 394–405. DOI: 10.1109/ICST62969.2025.10989041. URL: https://doi.ieeecomputersociety.org/10.1109/ICST62969.2025.10989041.