Otimização da Geração de Casos de Teste em Aplicativos Móveis via Aprendizado por Reforço e Agentes de IA

Mailton Viana Farias
UFAM/INdT
Manaus, Brasil
mvf2@icomp.ufam.edu.br

Mateus Luiz de Oliveira Souza
UFAM/INdT
Manaus, Brasil
souza.mateus@ufam.edu.br

Letícia Gabrielly B. dos Santos
UFAM
Manaus, Brasil
leticia.santos@icomp.ufam.edu.br

Ana Caroline N. Gomes
UFAM
Manaus, Brasil
ana.gomes@icomp.ufam.edu.br

Suelen da Silva Pereira
UFAM
Manaus, Brasil
suelen.pereira@icomp.ufam.edu.br

Eliane Collins
Instituto de Desenvolvimento Tecnológico (INdT)
Manaus, Brasil
eliane.collins@indt.org.br

Resumo

A necessidade de acelerar a entrega de software tem ampliado a busca por automação para aumentar a produtividade em garantia da qualidade. Nesse contexto, técnicas de Inteligência Artificial (IA), incluindo Modelos de Linguagem de Larga Escala (LLMs), oferecem novas possibilidades para a geração e o processamento de artefatos de teste. Entretanto, os logs brutos produzidos pela exploração baseada em DRL apresentam baixa estrutura semântica, dificultando sua interpretação e utilização em atividades de teste. Este trabalho apresenta uma extensão da DRL-MOBTEST que integra um LLM executado localmente para transformar logs de exploração em casos de teste estruturados conforme a ISO/IEC/IEEE 29119-3. A solução incorpora geração automática de requisitos, execução em duas fases, cobertura com JaCoCo, mecanismo Anti-Stuck e orquestração por interface de linha de comando, reduzindo intervenções manuais e a dependência de serviços de IA em nuvem. A abordagem foi avaliada em 15 aplicações Android de código aberto, com quatro rodadas de teste independentes por aplicação. Os resultados alcançaram 73,02% de cobertura média de requisitos, 41,90% de linhas, 47,52% de métodos e 26,30% de branches, além de taxa de sucesso da transcrição superior a 99%. A solução pode apoiar profissionais e pesquisadores na criação, execução e análise de testes de software, mantendo o processamento semântico em ambiente local.

Palavras-chave

Geração Automática de Casos de Teste, Aprendizado por Reforço Profundo, Testes em Aplicativos Móveis, Análise de Cobertura, Agentes Autônomos.

1 Introdução

A execução de testes por método de exploração em aplicações Android por técnicas de Aprendizado por Reforço Profundo (*Deep Reinforcement Learning* – DRL) tem se mostrado uma alternativa promissora para ampliar a cobertura funcional e reduzir o esforço manual na criação de casos de teste [1]. Nesse contexto, a ferramenta DRL-MOBTEST, proposta por Collins et al. [2], automatiza a navegação em interfaces gráficas (*Graphical User Interface* – GUI) por meio de um agente baseado em DRL, alcançando resultados superiores aos de abordagens tradicionais em estado da arte e de testes aleatórios.

Apesar desses avanços, a utilização da ferramenta em ambientes de mercado ainda enfrenta limitações práticas. A exploração nativa produz logs de baixo nível compostos por eventos, coordenadas e identificadores espaciais. Conforme apontado por Marques et al. [3], essa granularidade técnica dificulta a interpretação semântica e a reutilização direta dos artefatos por engenheiros de teste. Além disso, o processo carece de integração com métricas de cobertura de código e de mecanismos que favoreçam sua incorporação em *pipelines* de Garantia da Qualidade. Em cenários industriais, essas limitações são agravadas por requisitos de confidencialidade, que frequentemente restringem o envio de dados para serviços de Inteligência Artificial baseados em nuvem.

Marques et al. [3] avançaram na melhoria da legibilidade dos casos de teste gerados pela DRL-MOBTEST por meio do uso de Modelos de Linguagem de Larga Escala (*Large Language Models* – LLMs). Entretanto, a abordagem ainda exige intervenção manual durante o *pipeline*, não fornece rastreabilidade funcional integrada nem incorpora métricas de cobertura estrutural. Este trabalho amplia essa proposta ao apresentar uma extensão da DRL-MOBTEST que automatiza todo o fluxo pós-exploração por meio de um LLM executado localmente, preservando a privacidade dos dados e eliminando dependências de serviços externos.

A solução proposta integra quatro funcionalidades principais: (i) geração automática de casos de teste padronizados conforme a ISO/IEC/IEEE 29119-3 a partir dos logs produzidos pela exploração; (ii) coleta automática de métricas de cobertura utilizando JaCoCo (*Java Code Coverage*) com geração de relatórios HTML; (iii) uma interface de linha de comando (*Command-Line Interface* – CLI) que orquestra todo o *pipeline* de forma automatizada; e (iv) um mecanismo *Anti-Stuck*, responsável por detectar e recuperar automaticamente situações em que o agente permanece preso em estados repetitivos durante a exploração.

A solução foi avaliada em 15 aplicativos Android de código aberto, alcançando taxas de transcrição superior a 99%, cobertura média de 73,02% dos requisitos funcionais e geração automática de artefatos que consolidam casos de teste e métricas de cobertura. Os resultados demonstram que a extensão proposta amplia a interpretabilidade e a automação do processo de teste, fornecendo uma base robusta para apoiar testadores e pesquisadores da área de *Quality Assurance* (QA) na consolidação de fluxos de qualidade de software mais autônomos e seguros.

2 Solução Proposta

A arquitetura proposta automatiza o processamento dos artefatos gerados durante a exploração de aplicações Android. Preservando o mecanismo original baseado em DRL da DRL-MOBTEST [2], a solução introduz novas funcionalidades orquestradas integralmente por uma Interface de Linha de Comando (CLI), voltadas à adoção prática em processos de qualidade de software. A Figura 1 apresenta o *pipeline* dessa abordagem.

A extensão também substitui etapas anteriormente manuais por uma execução automatizada e reprodutível. A configuração de múltiplos aplicativos via `settings.json`, a geração automática dos requisitos e a execução sequencial das fases de exploração e treinamento permitem processar diferentes aplicações em um único fluxo, reduzindo intervenções entre etapas.

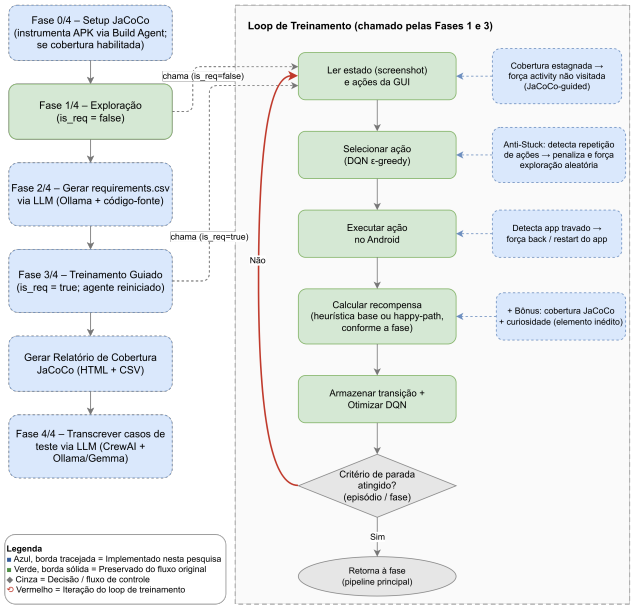

Figura 1: Fluxograma operacional do pipeline da solução proposta sobre o ecossistema DRL-MOBTEST.

Ciclo de Exploração e Requisitos. Conforme ilustrado na Figura 1, a arquitetura organiza a execução em duas fases sequenciais. Inicialmente, o agente DQN interage com a interface de forma não guiada para mapear a estrutura visual e os componentes da aplicação. Utilizando o repositório de código-fonte da aplicação como entrada obrigatória, o sistema cruza os elementos visuais mapeados com a análise estática para extrair requisitos funcionais orientados à interface e à navegação (*e.g.*, interações esperadas com componentes e transições de tela), consolidando-os no arquivo `requirements.csv`. Em seguida, uma nova fase de treinamento é executada, orientada por esses requisitos, para direcionar a exploração aos cenários identificados. Durante as etapas de execução, o mecanismo *Anti-Stuck* monitora o comportamento do agente e aciona estratégias de recuperação quando detecta estados repetitivos ou interrupções de interface, como "diálogos modais".

Processamento Semântico e Conformidade. Ao término da execução, o *pipeline* reúne automaticamente os logs de exploração, registros de *crashes* e métricas de cobertura estrutural obtidas por meio do JaCoCo. Esses artefatos são processados por um agente baseado em CrewAI [4], utilizando o modelo Gemma 3:4b executado localmente via Ollama, responsável por converter os logs em casos de teste estruturados conforme a ISO. Como resultado, a solução gera automaticamente casos de teste padronizados, relatórios HTML e arquivos CSV contendo as métricas de cobertura e os artefatos produzidos durante a execução. Para aumentar a robustez da exploração, o mecanismo *Anti-Stuck* identifica repetições de ações e estados de travamento da aplicação, acionando estratégias de recuperação sem intervenção manual.

Assim, a extensão preserva o núcleo de exploração da DRL-MOBTEST e concentra suas contribuições na automação do fluxo, na robustez da execução, na instrumentação de cobertura e na transformação dos resultados de baixo nível em artefatos de teste semanticamente estruturados.

3 Avaliação e Resultados

Configuração e Metodologia. A solução proposta foi avaliada em 15 aplicativos Android de código aberto provenientes do repositório F-Droid. Para garantir a reprodutibilidade, os experimentos foram conduzidos no sistema operacional Linux, utilizando uma estação de trabalho equipada com processador Ryzen 9 9900x, 64GB de memória RAM e GPU Nvidia RTX-5070Ti. A exploração interativa ocorreu em dois dispositivos físicos Motorola Edge 60 Pro executando o Android 15. Cada aplicativo foi submetido a quatro rodadas independentes, compostas sequencialmente por exploração autônoma de descoberta (1 hora) e treinamento guiado por requisitos (1 hora). A cobertura estrutural foi obtida automaticamente via JaCoCo, e a transcrição dos logs utilizou o modelo Gemma 3:4b executado localmente. A cobertura de requisitos considera um requisito coberto quando a execução registra uma ação correspondente à funcionalidade especificada, sendo calculada pela razão entre requisitos cobertos e requisitos identificados.

Resultados de Cobertura e Transcrição. A Tabela 1 apresenta as métricas obtidas para os 15 aplicativos avaliados.

Tabela 1: Métricas de Cobertura de Testes por Aplicativo

App	Requisitos	Instrução	Linhas	Branches	Métodos
Budget Watch	67,53%	51,86%	51,08%	26,07%	60,45%
Calculator You	90,45%	41,02%	46,48%	30,98%	52,75%
Catima	1,28%	44,17%	45,70%	31,43%	53,95%
Clock	53,08%	45,69%	47,75%	29,49%	55,72%
Contacts	89,78%	49,49%	48,25%	31,90%	38,30%
Exceer	66,00%	37,16%	50,90%	45,10%	57,30%
Food Records	100,00%	41,13%	40,25%	25,33%	40,38%
Gallery	80,40%	15,30%	14,00%	8,60%	16,40%
Habits	83,33%	36,87%	37,53%	22,55%	44,30%
Money Tracker	82,95%	51,86%	41,83%	24,43%	48,58%
OpenContacts	64,38%	25,22%	26,05%	14,00%	31,30%
Piggsy	82,73%	45,57%	37,00%	21,88%	41,43%
ShoppingList	85,18%	41,71%	41,08%	22,63%	51,08%
SiliNote	92,93%	69,37%	70,48%	42,08%	79,85%
ToDoList	55,33%	30,89%	30,18%	18,00%	40,98%

Em média, a abordagem alcançou 73,02% de cobertura de requisitos funcionais (mediana de 82,73%), além de coberturas estruturais

médias de 41,90% para linhas, 47,52% para métodos e 26,30% para *branches*. Esses resultados evidenciam a capacidade da extensão de produzir simultaneamente artefatos funcionais e métricas estruturais a partir da exploração automatizada. Algumas variações observadas decorrem de limitações conhecidas das *APIs* de automação de interface, como o *UIAutomator*, especialmente em aplicações que exigem permissões em tempo de execução ou utilizam componentes altamente dinâmicos, restringindo a exploração de determinados cenários.

A análise dos resultados também evidenciou os pré-requisitos de viabilidade da ferramenta. Para uma transcrição e exploração plenas, a solução exige que as aplicações exponham componentes padronizados acessíveis via *UIAutomator* e não exijam interações manuais para a liberação de permissões do sistema. Durante os experimentos, constatou-se que o aplicativo Catima é estruturalmente inviável para o atual *pipeline* por não cumprir esses requisitos de acessibilidade, resultando em uma cobertura funcional quase nula (1,28%), apesar de o agente ter alcançado 45,70% de cobertura de linhas às cegas. De forma semelhante, o aplicativo Gallery teve sua cobertura estrutural restrita (14,00%) por bloqueios em permissões de mídia. A manutenção e discussão desses cenários de exceção no estudo é fundamental, pois explicita as condições de contorno da abordagem e define o perfil de aplicativos compatíveis para a reprodutibilidade da ferramenta.

Impacto Prático e Aplicabilidade. O principal avanço operacional observado foi na etapa de transcrição semântica. Considerou-se bem-sucedida a transcrição que gerou automaticamente um caso de teste completo no formato esperado, sem necessidade de intervenção manual. Nessa configuração, a arquitetura obteve taxa de sucesso superior a 99%, convertendo logs brutos de interação em casos de teste legíveis, estruturados conforme a norma ISO/IEC/IEEE 29119-3. Essa automação mitiga a dependência de oráculos manuais para a leitura de artefatos de baixo nível, entregando aos engenheiros de teste uma documentação diretamente auditável integrada a relatórios HTML unificados. Dessa forma, a solução apresenta potencial para suporte à geração de casos de teste e processos de garantia da qualidade, reduzindo o esforço analítico sem expor dados sensíveis corporativos a serviços em nuvem. A extensão atua principalmente nas etapas de automação, processamento semântico e análise dos artefatos, não alterando o núcleo DQN nem seus requisitos computacionais.

4 Conclusão

Este trabalho apresentou uma extensão da DRL-MOBTEST para reduzir a lacuna semântica entre a exploração autônoma e a geração de artefatos de teste. A integração de um LLM executado localmente, do mecanismo *Anti-Stuck* e da automação do *pipeline* via CLI reduz o esforço manual de interpretação dos logs brutos e permite gerar artefatos estruturados sem expor dados de teste a serviços externos de IA.

A avaliação em 15 aplicativos Android indicou a viabilidade da abordagem, já que o agente alcançou 73,02% de cobertura média de requisitos, além de coberturas estruturais médias de 41,90% para linhas, 47,52% para métodos e 26,30% para *branches*. Na etapa de transcrição, a solução apresentou taxa de sucesso superior a 99%, convertendo interações de baixo nível em casos de teste estruturados conforme a ISO/IEC/IEEE 29119-3.

A análise dos resultados também evidenciou que as limitações observadas na cobertura estão associadas, em parte, às características das aplicações e da infraestrutura de automação Android, e não exclusivamente ao processamento semântico. *Frameworks* de testes de interface como o *UIAutomator* apresentam dificuldades diante de permissões restritivas e componentes altamente dinâmicos, limitando a exploração de determinados cenários.

Em conjunto, os resultados demonstram o potencial da solução para apoiar testadores, pesquisadores e processos de QA em ambientes industriais, ao oferecer um *pipeline* mais automatizado, estruturado e auditável. Contudo, reconhece-se que a avaliação atual ainda carece de testes em um conjunto mais amplo e diversificado de aplicativos para assegurar maior relevância estatística aos achados. Como evolução, trabalhos futuros podem focar na expansão da base de testes empíricos, bem como investigar a incorporação de modelos multimodais executados localmente, buscando ampliar a percepção da interface e contornar as restrições da automação baseada exclusivamente no *UIAutomator*.

Disponibilidade de Artefatos

Todos os artefatos produzidos e utilizados nesta pesquisa estão disponíveis publicamente para garantir a reprodutibilidade dos experimentos e fomentar investigações futuras. Os materiais podem ser acessados através do repositório https://figshare.com/s/f1d80cf8f543fe4df74b, e estão disponibilizados sob a licença CC-BY 4.0.

Agradecimentos

Agradecemos ao Instituto de Computação (IComp) da UFAM pelo apoio acadêmico mediante à estrutura da Especialização/Projeto IARTES e ao INDT pelo fomento financeiro para a apresentação deste artigo. Em conformidade com o Código de Conduta da SBC, declaramos o uso das Ias Generativas Gemini (Google) e ChatGPT (OpenAI) como assistentes restritos à revisão gramatical, fluidez, organização lógica e ajustes técnicos em LaTeX. Todas as sugestões foram revisadas e validadas pelos autores, que assumem integral responsabilidade pelo conteúdo final deste manuscrito.

Referências

[1] Ana Paula Cardoso, Cleicy Priscilla Santos, Eliane Collins, Kelen Lima, Pablo Quiroga, and Marlon Griego. 2023. Evaluation of Automatic Test Case Generation for the Android Operating System using Deep Reinforcement Learning. In *Proceedings of the XXII Brazilian Symposium on Software Quality* (Brasília, Brazil) *(SBQS '23)*. Association for Computing Machinery, New York, NY, USA, 228–235. doi:10.1145/3629479.3629503

[2] Eliane Collins, Arilo Neto, Auri Vincenzi, and José Maldonado. 2021. Deep Reinforcement Learning based Android Application GUI Testing. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering* (Joinville, Brazil) *(SBES '21)*. Association for Computing Machinery, New York, NY, USA, 186–194. doi:10.1145/3474624.3474634

[3] João Paulo Marques, Mônica Lima, Bruno Souza, Eloise Miranda, André Santos, and Eliane Collins. 2023. SelectNLTest - Selection and natural language rewriting of test cases generated by the DRL-MOBTEST tool. In *Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing* (Campo Grande, MS, Brazil) *(SAST '23)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3624032.3624043

[4] João Moura. 2024. CrewAI: Framework for orchestrating role-playing autonomous AI agents. https://github.com/joaomdmoura/crewAI. Acesso em: jun. 2026.