Automatização de Testes de Aplicativos Móveis com Aprendizado de Máquina por Reforço Utilizando Requisitos em Histórias de Usuário

Gabriele Guimarães[1], Suelen da Silva Pereira[1], Thiago Santos Borges[1], Nora Neyse Torres da Cunha[1], João Paulo Marques[2], Eliane Collins[2]

[1]Instituto de Computação – Universidade Federal do Amazonas – Manaus, AM – Brasil

[2]Instituto de Desenvolvimento Tecnologico – INDT – Manaus, AM – Brasil

gabriele.guimaraes@icomp.ufam.edu.br, suelen.pereira@icomp.ufam.edu.br, thiago.borges@icomp.ufam.edu.br, nora.cunha@icomp.ufam.edu.br, joao.marques@indt.org.br, eliane.collins@indt.org.br

Abstract. *Information systems are mainly accessed via mobile applications. Ensuring their quality is crucial, which is why software testing is essential. However, creating and executing manual tests requires a lot of effort. This study investigates the automation of test generation for Android applications, combining the reinforcement learning DRL-MOBTest tool with language models to interpret and structure requirements extracted from user stories written in Gherkin. In experiments conducted with 10 applications, the results indicated an average requirements coverage of 81.41%. Despite the good results, the solution has limitations for scenarios with more sequential interactions. These findings may contribute to research in the application validation using artificial intelligence.*

Resumo. *Os sistemas de informação são acessados principalmente via aplicativos móveis. Garantir a qualidade deles é crucial, por isso, testes de software são essenciais. Porém, a criação e execução de testes manuais demandam grande esforço. Este estudo investiga a automação da geração de testes para aplicativos Android, combinando a ferramenta de aprendizado de máquina por reforço DRL-MOBTest com modelos de linguagem para interpretar e estruturar requisitos extraídos de histórias de usuário escritas em Gherkin. Nos experimentos realizados com 10 aplicativos, os resultados indicaram uma cobertura média de requisitos de 81,41%. Apesar dos bons resultados, a solução possui limitações em cenários com mais interações sequenciais. Estes achados contribuem para pesquisas em validação de aplicações usando inteligência artificial.*

1. Introdução

Atualmente, a acessibilidade dos sistemas de informação por meio de aplicativos móveis é essencial, dado o aumento da demanda por praticidade. Porém, o mercado está cada vez mais exigente e intolerante a falhas, tornando indispensável o investimento em testes de software abrangentes. Embora os testes manuais sejam importantes, sua execução pode ser trabalhosa em projetos complexos e com prazos curtos. Nesse contexto, a automação

de testes surge como uma solução eficiente, especialmente com o uso de Inteligência Artificial (IA), que permite testes mais precisos e adaptáveis.

Este artigo propõe a utilização de grandes modelos de linguagem (GML), aliados ao aprendizado por reforço profundo, para gerar automaticamente casos de teste a partir de histórias de usuário escritas em Gherkin. A abordagem visa melhorar a qualidade do desenvolvimento de software, enfrentando os desafios da criação manual de testes e buscando cobrir os requisitos de forma ampla.

A solução envolve uma API que interpreta automaticamente as histórias de usuário em Gherkin com GML, converte essas informações em requisitos estruturados e executa os testes por meio da ferramenta DRL-MOBTest, que utiliza aprendizado por reforço para interagir com aplicativos. Testes realizados em 10 apps Android alcançaram uma cobertura média de requisitos de 81,41%. Assim, o estudo demonstra o potencial da IA na automação de testes, tornando o processo mais eficaz e acessível.

O artigo está dividido em: Seção 2, fundamentação teórica, Seção 3, descrição da solução proposta, Seção 4, experimentos realizados, Seção 5, os resultados e discussões e Seção 6 a conclusão.

2. Fundamentação Teórica

2.1. Histórias de Usuários e Gherkin

As histórias de usuário são amplamente utilizadas no desenvolvimento ágil para capturar requisitos de forma clara e objetiva [Cohn 2004]. Seguem um formato padronizado que facilita a comunicação entre equipes e stakeholders, auxiliando no desenvolvimento e priorização de funcionalidades. Essa abordagem tem sido amplamente adotada e demonstrou impacto positivo na clareza dos requisitos [Amna and Poels 2022].

O Gherkin é uma linguagem de domínio específico para a descrição estruturada de requisitos e comportamento do sistema. Sua sintaxe, baseada em palavras-chave como Given, When, Then, permite definir critérios de aceitação de forma clara, promovendo a automação de testes. [Härlin 2016].

2.2. Modelos LLaMA3 e DRL-MOBTest

GML são um tipo de modelo de IA criado para entender e gerar texto. Esses modelos são treinados com grandes volumes de dados de toda a internet; eles aprendem reconhecendo padrões sobre como as palavras e frases são comumente usadas juntas [Qin et al. 2024]. Neste trabalho, modelos abertos, como LLaMA3:8B, foram adotados para interpretar histórias de usuário e convertê-las em requisitos estruturados. Técnicas como quantização e fusão de modelos garantem eficiência no processamento. [Siriwardhana et al. 2024].

O aprendizado por reforço profundo (ARP) combina aprendizado por reforço com redes neurais profundas para otimizar a tomada de decisão em ambientes dinâmicos, sem necessidade de dados rotulados [Soares 2020]. Este trabalho utiliza o DRL-MOBTest, uma ferramenta de ARP para gerar e otimizar casos de teste em apps Android, através de exploração para criar testes e identificar falhas [Ribeiro 2022]. O módulo SelecNLTest complementa o DRL-MOBTest, refinando os testes, removendo redundâncias e convertendo-os para linguagem natural, facilitando sua interpretação pelos testadores. [Marques et al. 2023]. Essa abordagem reduz o esforço manual e melhora a documentação dos testes automatizados.

3. Solução Proposta

Este estudo consiste em complementar a ferramenta DRL-MOBTest utilizando requisitos estruturados em histórias de usuários de formato Gherkin, muito populares no desenvolvimento de software em sistemas de informação. Em seguida, apresenta-se a arquitetura desta proposta. (Figura 1) e uma descrição detalhada das etapas envolvidas na sua implementação.

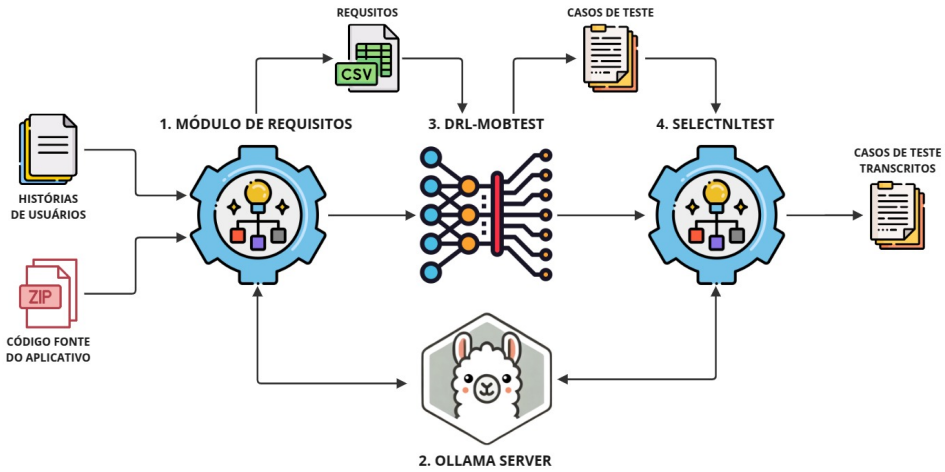

Figura 1. Arquitetura da Solução.

3.1. Extração e Processamento de Requisitos

No Módulo de Requisitos, o código-fonte do app é analisado para extrair informações como nome do pacote, activities e componentes de interface. Esses dados são organizados em um cabeçalho estruturado, para gerar o arquivo de requisitos compatível com o DRL-MOBTest. O código é tratado como um arquivo ZIP, permitindo a identificação e processamento do AndroidManifest.xml e dos arquivos XML que definem a interface gráfica.

Em seguida, os requisitos são extraídos de histórias de usuário e cenários Gherkin, armazenados em arquivos TXT. O modelo LLaMA3:8B interpreta essas histórias, organizando-as em um formato estruturado (Figura 2) para facilitar a geração de testes.

```
activity      field          id         action  type    size_start  size_end  value
app activity  button       id/menu       click   none
app activity  edittext  id/NameEdit     type    text       1          30     test
app activity  edittext  id/valueEdit    type    number     1           9      150
app activity  button    id/action_save  click   none
```

Figura 2. Formato do Arquivo de Requisitos para DRL-MOBTest.

3.2. Geração de Casos de Teste

O modelo LLaMA3:8B, executado via Servidor Ollama, gera um arquivo de requisitos compatível com o DRL-MOBTest, reduzindo custos e limitações de requisições. Esses

dados são convertidos em formato tabular com o Pandas. [Gupta and Bagchi 2024], e exportados para CSV.

O DRL-MOBTest possui o módulo de gerenciador de Agente que coleta dados de ações e estados, atribuindo recompensas quando o agente entra em uma nova tela, realiza ações que cobrem requisitos e menor recompensas ao repetir ações e sair da app. O módulo Ambiente executa os testes em dispositivos físicos ou emuladores, otimizando ações com ARP e registrando interações.

Para maior eficiência, aAPI com GML que elimina redundâncias nos testes, ampliando a cobertura dos requisitos. Além disso, os casos são documentados no padrão ISO/IEC/IEEE 2911, via LLaMA3:8B, garantindo clareza nas pré-condições, pós-condições, entradas e saídas esperadas.

4. Experimentos

Os experimentos foram conduzidos em uma infraestrutura composta por um processador Intel® Core™ i7-13700K (24 núcleos), 64 GiB de RAM, GPU NVIDIA GeForce RTX™ 3070 Ti, SSD de 2 TB e sistema operacional Ubuntu 24.04.1 LTS. Essa configuração garantiu um ambiente estável para a execução do DRL-MOBTest e do modelo LLaMA3:8B.

Foram avaliados 10 apps de código aberto do repositório FDroid [F-Droid.org 2020] que possuem diferentes configurações e elementos de interface. Foi utilizado o emulador Medium Phone API 30 com resolução de 1080 x 2400, integrado ao Android Studio como dispositivo de teste. Foram feitas quatro sessões de testes em cada app, com duração de duas horas, totalizando um tempo considerável para a validação. Essa abordagem possibilitou a confirmação das tendências nos resultados em cada execução [Ribeiro 2022].

Para a análise da cobertura de código, utilizou-se a biblioteca JaCoCo (Java Code Coverage), uma ferramenta [JaCoCo Team 2025], que fornece métricas de código incluindo cobertura de instruções (percentagem de instruções individuais de bytecode executadas), cobertura de ramos (percentagem de ramos de controle de fluxo, como `if/else` e `loops`, executados), cobertura de linhas (percentagem de linhas de código executáveis executadas) e cobertura de métodos (percentagem de métodos chamados), visando avaliar se os testes cobrem áreas do código que requerem atenção para garantir a confiabilidade do software.

A Tabela 1 apresenta informações detalhadas sobre os apps testados, incluindo tamanho, número de classes e outras características relevantes.

Para garantir a segurança das execuções, eram realizadas limpezas na memória do emulador, para assegurar a integridade dos dados do experimento.

5. Resultados e Discussões

Os experimentos realizados utilizaram histórias de usuário como entrada para a ferramenta DRL-MOBTest. Os resultados sugerem que a execução no modo requisito pode auxiliar profissionais de teste na geração automática de casos de teste para apps Android, uma vez que requisitos simples conseguem ser cobertos por testes, tornando o processo mais eficiente e alinhado aos requisitos do software.

Aplicativo	Versão	Instruções	Ramos	Linhas	Métodos
A Time Tracker	1.0	8319	531	1689	193
Daily Pill	1.0	2012	118	435	98
Exceer	0.2.3	5488	426	1112	216
Loaned	1.0.1	9572	490	1985	333
Money Tracker	2.1.3	13898	907	3323	733
openWorkout	2.3.0	23719	1506	5070	968
SiliNote	1.0	2342	60	528	133
Simple Text Editor	1.9.4	3233	210	788	161
To Do List	1.1	4316	213	522	104
Tricky Tripper	1.6.2	36617	2494	8400	1733

Tabela 1. Informações sobre os 10 aplicativos

5.1. Análise de Cobertura de Código

A análise da cobertura de código dos aplicativos testados revela variações nas métricas, conforme mostrado na Tabela 2. Os resultados indicam que a ferramenta opera conforme esperado, gerando testes válidos, mas com eficácia variável entre os aplicativos. Aplicativos como Simple Text Editor (86,38%) e Daily Pill (81,31%) apresentaram altos índices de cobertura, sugerindo que a abordagem funciona bem para requisitos mais diretos. No entanto, aplicativos como Exceer (47,68%) e Money Tracker (52,20%) tiveram coberturas mais baixas, possivelmente devido a requisitos mais complexos ou que exigem maior interação.

Aplicativo	Instruções	Ramos	Linhas	Métodos
A Time Tracker	67.45%	54.33%	65.42%	70.34%
Daily Pill	81.31%	61.23%	81.21%	80.87%
Exceer	47.68%	36.56%	51.28%	63.66%
Loaned	72.27%	54.23%	72.89%	83.10%
Money Tracker	52.20%	34.15%	50.77%	58.32%
openWorkout	53.01%	30.35%	56.00%	62.29%
SiliNote	69.09%	36.25%	68.28%	78.20%
Simple Text Editor	86.38%	74.52%	86.10%	86.18%
To Do List	68.30%	44.01%	77.20%	83.41%
Tricky	56.31%	40.89%	57.48%	63.29%

Tabela 2. Métricas de Desempenho dos Aplicativos

5.2. Análise de Cobertura de Requisitos

A análise foi conduzida por quatro pesquisadores que realizaram uma comparação manual entre as histórias de usuário e os casos de teste gerados. Inicialmente, o DRL-MOBTest gerou 31.641 casos de teste. Após a eliminação das redundâncias com SelectNTest, o número foi reduzido para 5.263 casos. Esse processo de transcrição contribuiu para aprimorar a legibilidade e a interpretação dos dados, tornando o procedimento mais eficiente.

A média de cobertura de requisitos foi calculada entre os apps testados verificando cada caso de teste criado se estava contemplando cenários de acordo com os requisitos.

O resultado foi de 81,41%, indicando uma tendência positiva na geração de testes a partir de requisitos. Os aplicativos openWorkout (92,59%) e Tricky (93,18%) apresentaram as maiores coberturas, enquanto Excel teve o menor valor (65,79%), conforme detalhado na Figura 3.

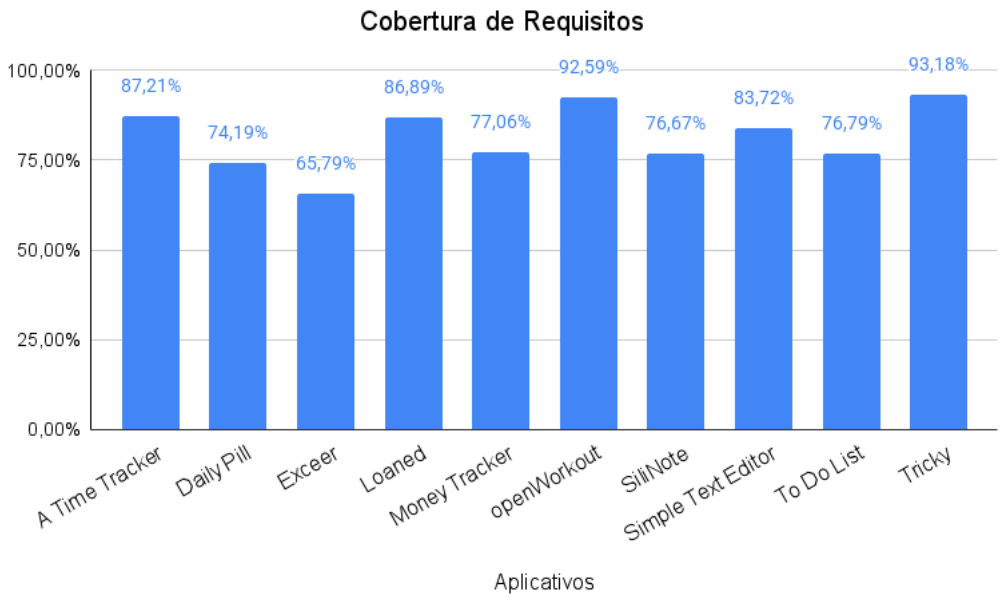

Figura 3. Cobertura de Requisitos

Apesar dos bons resultados, a cobertura foi afetada por limitações de desempenho do algoritmo ARP, como paradas prolongadas e interações demoradas, em que a ferramenta ficava presa em uma única tela ou alternava entre telas sem executar testes efetivos.

Requisitos simples foram totalmente cobertos, enquanto aqueles com mais interações ou desvios apresentaram maior dificuldade. Isso sugere a necessidade de melhor infraestrutura de hardware para maior velocidade de execução e aprimoramento do algoritmo de ARP para lidar com ações mais complexas. Essas limitações destacam a importância de melhorias futuras para aumentar a eficiência e a cobertura dos testes.

6. Conclusões e Trabalhos Futuros

O estudo demonstrou que a automação da geração de testes para aplicativos Android, utilizando o DRL-MOBTest e histórias de usuário, apresentou uma tendência promissora, atingindo uma cobertura média de 81,41% dos requisitos. No entanto, foram identificadas limitações, como interações prolongadas que impactaram a cobertura em alguns aplicativos, exigindo revisões manuais para ajuste dos resultados.

Futuras melhorias devem focar na eficiência da ferramenta e na precisão da extração de informações, além do aprimoramento do agente de teste para ampliar a cobertura e se aproximar de 100%.

Referências

Amna, A. R. and Poels, G. (2022). Systematic literature mapping of user story research. *IEEE Access*, 10:51723–51739.

Cohn, M. (2004). *Effective User Stories*, page 208–208. Springer Berlin Heidelberg.

F-Droid.org (2020). Free and open source android app repository.

Gupta, P. and Bagchi, A. (2024). *Introduction to Pandas*, pages 161–196. Springer Nature Switzerland, Cham.

Härlin, M. (2016). Testing and gherkin in agile projects. Master's thesis, Linköping University, Linköping, Sweden.

JaCoCo Team (2025). Jacoco - java code coverage library.

Marques, J. P., Lima, M., Souza, B., Miranda, E., Santos, A., and Collins, E. (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 (SAST '23)*, pages 77–85, Campo Grande, MS, Brazil. ACM.

Qin, M., Chang, D., Fischer, G., et al. (2024). The uniqueness of llama3-70b series with per-channel quantization. *Western Digital Research*.

Ribeiro, E. F. C. (2022). *DeepRLGUIMAT: Abordagem de Aprendizado de Máquina Profundo por Reforço Aplicado a Testes de GUI de Aplicações Móveis*. Tese de doutorado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, Brasil. Orientador: Prof. Dr. José Carlos Maldonado; Coorientador: Prof. Dr. Arilo Dias-Neto.

Siriwardhana, S., McQuade, M., Gauthier, T., Atkins, L., Fernandes Neto, F., Meyers, L., Vij, A., Odenthal, T., Goddard, C., MacCarthy, M., and Solawetz, J. (2024). Domain adaptation of llama3-70b-instruct through continual pre-training and model merging: A comprehensive evaluation. *Arcee*.

Soares, Y. C. P. (2020). Deep reinforcement learning for bipedal locomotion. *Universidade Estadual de Campinas*.