Aplicação Local e Open-Source de Grandes Modelos de Linguagem na Geração de Testes de Unidade

Gabriel Tonioni Duarte
Departamendo de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Brazil
gabriel.tonioni@dcc.ufmg.br

Marco Tulio Valente
Departamendo de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Brazil
mtov@dcc.ufmg.br

***Abstract*—O presente trabalho explora a viabilidade da adaptação do sistema Testpilot para utilizar um LLM de código aberto acessível localmente, com o objetivo de agilizar e facilitar o processo de teste de unidade. A abordagem utiliza o LM Studio para acessar um servidor local que executa o modelo open-source Llama 3.1 8B com quantização de 4 bits. São criados conjuntos de testes de unidade para 22 pacotes npm e os resultados são comparados com o Testpilot original em termos de métricas de cobertura. Os resultados mostram que não há diferença estatisticamente significativa entre a cobertura de statements e branches entre os dois experimentos, demonstrando a viabilidade da utilização de um LLM open-source acessível localmente, mesmo com restrições de espaço.**

***Index Terms*—Index Terms: Testes de unidade, Automação de testes, Grandes Modelos de Linguagem (LLMs), Open-Source, Llama, Testpilot.**

I. Introdução

Os testes de unidade são uma prática fundamental no desenvolvimento de software, que envolve a verificação do comportamento individual de pequenas partes de um programa, chamadas unidades. Eles são cruciais para garantir a correção do código, facilitando a detecção precoce de erros e permitindo que os desenvolvedores façam alterações com mais confiança.

A crescente complexidade do software moderno e a necessidade de entregas rápidas tornaram a escrita de testes de unidade uma obrigação dos desenvolvedores ao longo das últimas décadas. Portanto, a discussão atual sobre testes de unidade passa pelo aprimoramento das técnicas e pelo avanço da tecnologia para criação de testes de unidade [1]. Um dos avanços estudados no momento é a automação da criação de conjuntos de testes de unidade.

O benefício mais imediato da automação é a economia de tempo e esforço na criação de testes de unidade. Dessa forma, a automação da criação de conjuntos de testes de unidade se torna importante, pois, em um cenário ideal, um time de desenvolvedores consegue extrair o benefício máximo de um conjunto de testes com um esforço mínimo na sua criação e manutenção [1]. Ainda com relação ao tempo e esforço, o ponto negativo da adoção de testes de unidade no desenvolvimento de um sistema de software é o maior tempo de desenvolvimento gasto inicialmente para a criação dos testes, quando comparado com o desenvolvimento do sistema diretamente. Porém, esse ponto negativo é pago a longo prazo ao permitir o crescimento sustentável do sistema e ao permitir sua escalabilidade [1]. A automação da criação de conjuntos de testes de unidade também é importante por ajudar na mitigação desse ponto negativo.

Várias técnicas de automação de teste de unidade foram desenvolvidas, incluindo abordagens baseadas em análise estática e dinâmica de código. No entanto, a recente aplicação de Grandes Modelos de Linguagem (LLMs) para a geração automatizada de testes de unidade tem mostrado resultados promissores, com potencial para superar as limitações das técnicas anteriores, principalmente por meio do alcance de melhores métricas de cobertura e por gerar testes com melhor legibilidade [2].

O Testpilot, um sistema que utiliza LLMs para gerar testes de unidade automaticamente, foi avaliado em um estudo empírico [2]. Este estudo demonstrou a capacidade dos LLMs de gerar testes de unidade legíveis e que alcançam métricas de cobertura mais altas que outras ferramentas que não utilizam LLMs, como o Nessie. Os resultados alcançados pelo Testpilot são promissores. Porém, são apenas um primeiro passo em direção à evolução tecnológica da automação da criação de conjuntos de testes de unidade. A geração de muitos testes não é garantia de que o objetivo de um conjunto de testes de unidade foi atingido. Mesmo que os testes sejam sintaticamente corretos, legíveis e alcancem boas métricas de cobertura, como é o caso dos testes gerados pelo Testpilot, eles podem ser testes de má qualidade. Nesse caso, esse conjunto de testes não fornecerá um suporte adequado para o crescimento sustentável e a escalabilidade do sistema de software, levando o crescimento do sistema a um ponto de estagnação a longo prazo pelo aumento da complexidade e desordem, inclusive do código dos testes [1].

O Testpilot demonstradamente consegue dominar aspectos técnicos da escrita de testes de unidade, como a utilização de frameworks de teste (*e.g.* o framework Mocha). Porém,

o domínio completo da escrita de testes requer ir além do domínio técnico. Essencialmente, é preciso conseguir fazer um julgamento se um determinado teste é de boa qualidade ou de má qualidade. A capacidade de fazer esse julgamento corretamente requer um conhecimento de princípios, das melhores técnicas e dos padrões e antipadrões de escrita de testes. Ou seja, para contribuir para a evolução tecnológica dos testes de unidade na direção da automação da criação de conjuntos de testes de unidade, o Testpilot precisa ir além do domínio técnico. O domínio técnico da escrita de testes de unidade é alcançado a partir do desenvolvimento de técnicas para integrar um LLM em um processo de criação de testes de unidade. Esse processo de criação envolve técnicas para criação do prompt, processamento da resposta do modelo para criar um teste sintaticamente correto e análises de aspectos da qualidade do teste criado, como trivialidade do teste, utilização de asserts, estrutura do teste e legibilidade. Já a evolução a partir do domínio técnico depende principalmente do desenvolvimento e disponibilização de novos LLMs.

O Testpilot implementa uma técnica de geração adaptativa [2], [3] de testes de unidade e fornece uma sólida base de exploração do problema de automação da criação de conjuntos de testes de unidade. Os resultados iniciais do Testpilot também servem como referência para comparação com outros experimentos.

Este trabalho explora a viabilidade de adaptar o sistema Testpilot para utilizar um LLM de código aberto e acessado localmente. A principal motivação por trás dessa adaptação é agilizar e facilitar o processo de experimentação, particularmente à luz do surgimento contínuo e da rápida evolução de novos modelos.

Ao utilizar um LLM local, o sistema Testpilot ganha a capacidade de avaliar e integrar rapidamente novos modelos à medida que eles se tornam disponíveis, sem a necessidade de comunicação externa extensa ou dependência de serviços de terceiros. Essa abordagem localizada oferece várias vantagens distintas. Em primeiro lugar, ela fornece configurabilidade e controle completos sobre o LLM, permitindo ajustes e otimizações refinadas de acordo com os requisitos de experimentos específicos.

O uso de um LLM local também pode levar a uma economia de custos. Os pesquisadores podem evitar os custos associados à transferência de dados e taxas de uso que são incorridas ao acesso dos LLMs via uma API paga.

Ainda sobre o acesso local a LLMs, o framework de inferência de LLMs llama.cpp[1] é amplamente utilizado para execução local de LLMs. O llama.cpp utiliza o formato de arquivo GGUF[2], projetado para ser utilizado pela biblioteca de tensores GGML[3]. Um ponto importante sobre o formato GGUF é que ele fornece suporte a uma ampla variedade de métodos de quantização. Até mesmo uma quantização de 4 bits é suportada. Essa quantização mais agressiva é especialmente útil para a execução local de modelos em GPUs de consumo, muitas das quais possuem 8 GB de memória VRAM ou menos.

Outro ponto positivo do llama.cpp é o ecossistema de ferramentas que surgiu ao seu entorno. Isso facilita a adequação do framework aos requisitos de diferentes experimentos. Uma dessas ferramentas é o LM Studio[4]. O LM Studio implementa uma interface gráfica a partir da qual é possível pesquisar um catálogo de LLMs open-source, baixar o arquivo GGUF de um LLM, carregar o LLM totalmente na memória VRAM da GPU ou parcialmente na memória VRAM da GPU e parcialmente na memória RAM do sistema, configurar parâmetros de carregamento e configurar parâmetros de inferência. O LM Studio também oferece um chat para interação com o LLM carregado ou a execução de um servidor local que permite acesso ao LLM carregado através de uma API. O servidor local do LM Studio oferece a opção de uma API que segue o mesmo padrão da API utilizada para comunicação com os LLMs da Open AI[5]. Essa opção facilita o desenvolvimento de aplicações interoperáveis, que podem ser utilizadas, sem necessidade de alteração do código, com LLMs proprietários acessados remotamente ou com LLMs open-source acessados localmente.

Com a ajuda dessas tecnologias, este trabalho analisa a viabilidade das técnicas de escrita de testes de unidade do Testpilot com a utilização do LLM open source Llama 3.1 8B[6]. O Llama 3.1 8B é servido localmente pelo LM Studio.

A criação de um conjunto de testes de unidade para 22 pacotes npm, seguindo a abordagem do Testpilot e acessando o Llama 3.1 8B localmente, resultou em uma mediana de 37.5% de testes que passaram, uma mediana de 66.7% de cobertura de statements e uma mediana de 50.0% de cobertura de branches. Os respectivos valores de mediana alcançados pelo Testpilot são 48.0%, 70.2% e 53.2%, não muito distantes dos valores alcançados pelo experimento deste trabalho.

As seções subsequentes deste relatório apresentam uma discussão da abordagem utilizada, seguida da apresentação e análise dos resultados. O relatório termina com uma discussão sobre as implicações dos achados e a conclusão.

II. Abordagem

Este trabalho estende as avaliações originalmente feitas por Schäfer *et al.* [2], fazendo a geração de testes com um LLM open source acessado localmente. Com este intuito em mente, empregamos o LM Studio para ter acesso a um servidor rodando em uma máquina local. O LM Studio possui um catálogo de modelos para escolha. Todos os modelos do catálogo são open-source e cada modelo está disponível em diferentes níveis de quantização. O modelo escolhido no catálogo do LM Studio é o meta-llama-3.1-8b-instruct na sua versão com quantização de 4 bits. Isso permite o armazenamento do modelo por inteiro nos 8 GB de VRAM da GPU GTX 1070, utilizada na máquina local, no tempo de inferência.

[1]https://github.com/ggerganov/llama.cpp
[2]https://huggingface.co/docs/hub/gguf
[3]https://github.com/ggerganov/ggml
[4]https://lmstudio.ai/
[5]https://openai.com/
[6]https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

A geração dos testes neste trabalho segue a mesma abordagem utilizada por Schäfer *et al.* [2]. O sistema Testpilot gera um conjunto de testes completo para um pacote npm de interesse. Para cada método ou função exportada pelo pacote, são feitas no máximo 5 tentativas para gerar um teste que exercite a funcionalidade do método ou função da API do pacote. O Testpilot se comunica com o LLM através de uma API que segue o padrão estabelecido pela Open AI para a comunicação com os seus modelos proprietários. Especificamente, o Testpilot utiliza o endpoint /completions para gerar os testes. O endpoint /completions leva o modelo a prever os próximos tokens dado um prompt. O prompt construído pelo Testpilot e posteriormente enviado ao modelo contém o começo do código de um caso de teste. Então, o modelo completa o corpo do caso de teste, contendo o código do teste em si. Na construção do prompt, o Testpilot fornece informações contextuais ao modelo através de comentários, localizados antes do código do conjunto de testes.

O ponto de diferenciação deste trabalho está na utilização com acesso local do modelo open-source meta-llama-3.1-8b-instruct. A execução do modelo localmente em hardware de consumo, ou seja, utilizado pela população no geral ao contrário de ser um hardware especializado para o processamento de modelos de AI, gera restrições. A restrição mais importante é a restrição de espaço, determinada pela quantidade de memória VRAM da GPU. A restrição de espaço impõe a utilização de modelos com uma quantidade menor de parâmetros e com uma precisão menor dos pesos.

Este trabalho busca entender o impacto da restrição de espaço através da comparação de métricas de cobertura de statements e de cobertura de branches. A análise dessas métricas fundamentais permite entender a viabilidade do desenvolvimento de experimentos com LLMs para geração de testes em cenários com restrições próximas às restrições do ambiente de experimentação deste trabalho.

Para esta comparação inicial, não são feitas análises a respeito das características dos testes gerados, como diversidade, trivialidade dos testes e utilização de asserts. Esse tipo de caracterizações foi feito por Schäfer *et al.* [2] e por Siddiq *et al.* [4]. Este não é o foco do trabalho. O foco está na avaliação do impacto mais imediato nas métricas de cobertura gerado pelas restrições de executar o LLM localmente.

III. Resultados

A tabela 1 mostra uma comparação entre o experimento deste trabalho, chamado de Testpilot Modificado na tabela, e o experimento original, chamado de Testpilot Original na tabela. São comparados entre os dois experimentos o número de testes gerados, quantos dos testes gerados passaram, a porcentagem dos testes gerados que passaram, a cobertura de statements e a cobertura de branches.

Nas colunas de cobertura de statements e cobertura de branches estão marcados em azul os pacotes para os quais o Testpilot Modificado teve um valor de cobertura maior que o Testpilot Original. Na última linha da tabela temos a informação da mediana para as três métricas de desempenho de cada experimento.

Ao fazer um teste dos postos sinalizados de Wilcoxon[7] entre as medidas de cobertura de statements, o p-value encontrado foi 0.32. Isso indica que não há diferença estatisticamente significativa entre a cobertura de statements dos experimentos. O teste dos postos sinalizados de Wilcoxon entre as medidas de cobertura de branches gerou um p-value de 0.72. Então, entre as medidas de cobertura de branches dos experimentos também não há diferença estatisticamente significativa.

O gráfico 1 é um gráfico de barras agrupadas que mostra a diferença entre os valores das métricas do Testpilot Modificado em relação às métricas do Testpilot Original. A partir do gráfico é possível perceber que o Testpilot Modificado apresenta maiores discrepâncias no número total de testes gerados em alguns dos pacotes, como o memfs e o omnitool, e no número total de testes que passaram no de caso de alguns dos pacotes, como o memfs.

IV. Discussão

Os resultados das métricas de cobertura mostrados na tabela evidenciam a viabilidade da utilização de um LLM open-source acessado localmente. A quantização mais agressiva de 4 bits, necessária para conseguir armazenar o meta-llama-3.1-8b-instruct localmente, e o menor tamanho do modelo, com 8 bilhões de parâmetros, não impactaram os valores das métricas de cobertura o suficiente para causar uma diferença estatisticamente significativa em relação aos valores das métricas do experimento original. A partir dessa garantia inicial, se torna mais seguro prosseguir com outros tipos de experimentos.

As maiores discrepâncias indicadas na figura com relação ao número total de testes gerados e com o número de testes que passaram possivelmente tem relação com a menor quantidade de parâmetros do modelo e com a quantização mais agressiva utilizada.

V. Conclusão

Este relatório relata um estudo que explora a viabilidade de adaptar o sistema Testpilot, que usa LLMs para gerar testes de unidade automaticamente, para utilizar um LLM open-source e acessado localmente.

A principal motivação é agilizar e facilitar o processo de experimentação, particularmente à luz do surgimento contínuo e da rápida evolução de novos modelos.

Os resultados da modificação feita no experimento deste trabalho, que diz respeito ao acesso do Llama 3.1 8B localmente, foram comparados com os resultados alcancados por Schäfer *et al.* [2]. Os resultados deste trabalho mostram que a quantização de 4 bits e o menor tamanho do modelo não impactaram significativamente os valores das métricas de cobertura.

Portanto, a utilização de um LLM open-source acessado localmente é viável para a condução de experimentos.

[7]https://www.kaggle.com/discussions/general/420273

	Testpilot Modificado (meta-llama-3.1-8b-instruct)					*Testpilot Original (gpt3.5-turbo)*				
Nome do Pacote	**Número de Testes**	**Testes que Passaram**	**Testes que Passaram (%)**	**Cobertura de Statements (%)**	**Cobertura de Branches (%)**	**Número de Testes**	**Testes que Passaram**	**Testes que Passaram (%)**	**Cobertura de Statements (%)**	**Cobertura de Branches (%)**
bluebird	313	101	32	55.7	39.3	370	204	55.2	68.0	50.0
complex.js	137	96	70	73.4	50.1	209	121	58.0	70.2	46.5
countries-and-timezones	29	25	86	99.1	78.7	28	13	46.4	93.1	69.1
crawler-url-parser	14	4	29	75.7	53.8	14	2	14.3	51.4	35.0
dirty	92	39	42	72.6	65.4	70	32	45.3	74.5	65.4
fs-extra	553	301	54	57.5	37.6	471	277	58.8	58.8	38.9
geo-point	72	37	51	89.0	70.6	76	50	65.8	87.8	70.6
gitlab-js	51	20	39	66.7	50.0	141	14	9.9	51.7	16.5
image-downloader	5	1	20	66.7	50.0	5	4	80.0	63.6	50.0
js-sdsl	497	100	20	42.0	46.1	409	46	11.3	33.9	24.3
jsonfile	19	13	68	76.6	67.6	13	6	48.0	38.3	29.4
memfs	1264	164	13	60.5	41.6	1037	471	45.4	81.1	58.9
node-dir	32	18	56	72.7	62.3	40	19	48.1	64.3	50.8
omnitool	856	227	27	58.1	38.8	1033	330	31.9	74.2	55.2
plural	14	7	50	69.2	50.0	13	8	61.5	73.8	59.1
pull-stream	92	1	1	18.9	0.0	83	34	41.0	69.1	52.8
q	403	144	36	66.0	52.9	323	186	57.6	70.4	53.7
quill-delta	178	26	15	46.9	36.3	152	33	21.7	73.0	64.3
rsvp	116	42	36	64.9	51.5	109	70	64.2	70.1	55.3
simple-statistics	224	133	59	78.0	59.3	353	250	70.9	87.8	71.3
uneval	8	0	0	0.0	0.0	7	2	28.6	68.8	58.3
zip-a-folder	16	9	56	75.9	61.9	11	6	54.5	84.0	50.0
Mediana			**37.5**	**66.7**	**50.0**			**48.0**	**70.2**	**53.2**

TABELA 1: Comparação do número de testes gerados, número de testes que passaram, porcentagem de testes que passaram, cobertura de statements e cobertura de branches entre o experimento deste trabalho, utilizando o modelo meta-llama-3.1-8b-instruct, e o experimento feito por Schäfer *et al.* [2], utilizando o modelo gpt3.5-turbo, para 22 pacotes npm.

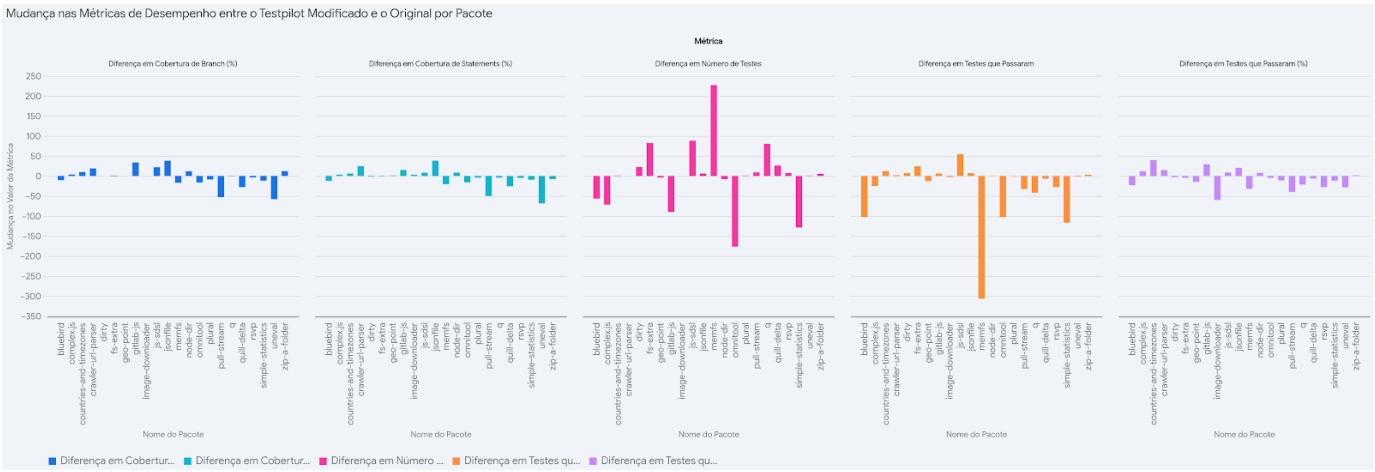

GRÁFICO 1: gráfico de barras agrupadas mostrando a magnitude das diferenças dos valores do experimento deste trabalho em relação aos valores do experimento feito por Schäfer *et al.* [2] para cada um dos 22 pacotes npm e cada uma das métricas.

REFERENCES

[1] V. Khorikov, "Unit testing: Principles, practices and patterns," 2020.

[2] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," *IEEE Transactions on Software Engineering*, vol. 50, no. 1, pp. 85–105, 2024.

[3] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, "Chatunitest: A framework for llm-based test generation," 2024.

[4] M. L. Siddiq, J. C. S. Santos, R. H. Tanvir, N. Ulfat, F. A. Rifat, and V. C. Lopes, "Using large language models to generate junit tests: An empirical study," 2024.