Paralelismo aplicado na comparação de descrições de assert de testes unitários

Vítor Augusto Ueno Otto
vitoruenootto@gmail.com
Instituto Federal Catarinense Campus Blumenau
Blumenau, Santa Catarina, Brasil

Abstract

O presente artigo apresenta um algoritmo de análise de similaridade de mensagens de descrição de erro de asserts em testes unitários gerados por uma ferramenta de desenvolvimento próprio e as LLMs (Large Language Models) ChatGPT e Gemini, bem como os ganhos obtidos através do uso de técnicas multi-thread neste algoritmo. Os resultados indicam uma similaridade média de 71% para o ChatGPT e 74% para o Gemini, sugerindo que a ferramenta pode ser uma boa alternativa para a geração de mensagens de erro de assert, mas com a vantagem de ser mais rápido, apresentar resultados mais constantes e menos erros de compilação. Quanto ao uso das técnicas de multi-thread, obteve-se um ganho de até 7,53 vezes com 8 threads em relação à execução sequencial do algoritmo.

Keywords

Testes unitários, Assertion Roulette, Large Language Models, Multithread

1 Introdução

O uso de testes de unidade como uma forma de aferir e aprimorar a qualidade de software é uma prática que vem ganhanado aderência no mercado [3]. Apesar disso, a qualidade dos testes de unidade também deve ser levada em consideração, do contrário, a manutenção do código de produção pode se tornar difícil e confusa [6].

Na literatura, dentro dos test smells, classificação de más práticas em testes de unidade, encontra-se o assertion roulette, que ocorre quando um caso de teste possui múltiplos asserts (afirmações) sem uma mensagem descritiva, que auxilia o programador em caso de falha no teste [8]. Apesar de ser um dos test smells que mais afetam a legibilidade do código [1], existem poucas ferramentas que realizam tanto a identificação quanto a refatoração do assertion roulette [2][7].

Para superar essas limitações, uma ferramenta foi criada como um trabalho de conclusão de curso. Essa ferramenta gera mensagens descritivas para asserts em testes unitários de forma automática com base no tipo de assert e nos dados envolvidos na afirmação. A ferramenta é compatível com asserts em JUnit 4 e 5.

Para validar a eficácia da ferramenta, foram realizados experimentos comparativos de similaridade entre as mensagens geradas por ela e aquelas produzidas por LLMs (Large Language Models, ou Grandes Modelos de Linguagem), como ChatGPT e Gemini. Durante esses experimentos, observou-se que o algoritmo de comparação apresentava um tempo de execução elevado, evidenciando a necessidade de melhorias de desempenho. Essa limitação destacou uma oportunidade para implementar técnicas de multi-threading, visando otimizar a execução e reduzir significativamente o tempo de processamento.

1.1 Objetivos

O presente trabalho tem como objetivo apresentar os resultados da análise de similaridade entre as mensagens geradas pela ferramenta de descrição de asserts e aquelas produzidas por LLMs (Large Language Models), como ChatGPT e Gemini. Além disso, busca destacar os ganhos de desempenho obtidos com a aplicação de técnicas de multi-threading, que foram implementadas para otimizar o tempo de execução do algoritmo comparativo. A análise também aborda os possíveis prós e contras da ferramenta, bem como possíveis pontos de melhoria e trabalhos futuros.’

2 Metodologia

O experimento consistiu em duas partes: a comparação de similaridade das mensagens de descrição de assert e a medida de desempenho para execuções com 1, 4 e 8 threads.

Para a primeira etapa do experimento, as LLMs selecionadas foram o ChatGPT[1] e o Gemini[2] por sua relevância e utilização em outros trabalhos [9]. Apesar disso, foram realizadas tentativas com o llama[3], modelo gratuito, mas que se mostrou lento demais para os testes. De forma mais específica, os modelos utilizados foram o gpt-3.5-turbo e o gemini-1.5-flash, por serem duas alternativas populares e de propósito geral.

Utilizou-se de práticas comuns elaboração de prompts utilizadas na literatura para interagir com as LLMs [9] [4]. Esta consisite em montar o prompt em duas partes, uma que contextualiza a LLM quanto ao cenário e ao papel que ela deve assumir (role ou contexto), e outra que instrui ela quanto ao que deve fazer de forma clara e as expectativas da tarefa (instrução) [9]. A string de contextualização e a base da mensagem de instrução estão exibidos na Figura 1. Note que a mensagem de instrução não está completa porque durante a execução os asserts são concatenados para formar a mensagem final.

Para a comparação entre as mensagens geradas pela ferramenta com as geradas pelas LLMs, utilizou-se do algoritmo Jaro de similaridade de strings por sua capacidade de gerar scores com

[1] https://openai.com/index/chatgpt/
[2] https://gemini.google.com/?hl=pt-BR
[3] https://www.llama.com/

qualidade [5]. O resultado do algoritmo Jaro é uma numeração entre 0 e 1 que indica o quão similar duas strings são, com 0 indicando que as strings não são nem um pouco similares e 1 indicando serem idênticas. Optou-se pelo uso de uma implementação pronta do algoritmo, disponível através da biblioteca python jaro-winkler [4].

Já para a segunda parte do experimento, a análise de desempenho da ferramenta considerou a média de tempo de três execuções, levando em conta os 10 primeiros asserts de cada um dos projetos analisados. Ao todo, foram utilizados os valores de uma, quatro e oito threads, limitadas pelo número de núcleos do computador do autor. Apenas três execuções foram realizadas por limitações de tempo, tendo em vista que para algumas das execuções leva-se mais de 20 minutos para o fim do experimento. Além disso, a escolha do uso de 10 asserts por projeto se deu por limitações do número de requisições diárias permitidas pelas LLMs[5].

Ao todo, foram considerados 32 projetos para o experimento, que tiveram como critério de seleção utilizarem Java e Junit, serem open source e possuirem uma quantidade razoável de asserts. A lista de projetos utilizados está presente na Tabela 1. Não obstante, foram levados em consideração apenas os tipos de assert que a ferramenta geradora de descrições suporta, que estão presentes na Tabela 2.

Table 1: Projetos Open Source

#	Projeto	Versão
1	Avro	1.11.1
2	Cucumber JVM	7.11.1
3	Disruptor	4.0.0.RC1
4	Dropwizard	2.1.4
5	Fastjson2	2.0.24
6	GraphHopper	6.2
7	Jasypt Spring Boot	3.0.5
8	JavaParser	3.25.1
9	JDA	5.0.0-beta.5
10	JetCache	2.7.3
11	JFreeChart	1.5.3
12	Jodd	5.3.0
13	Jsoup	1.15.3
14	Liquibase	4.19.1
15	Logback	1.4.5
16	OpenPDF	1.3.30
17	OptaPlanner	9.35.0.Beta2
18	POI-TL	1.12.1
19	Recaf	2.21.13
20	Redisson	3.19.3
21	RipMe	1.7.95
22	RoaringBitmap	0.9.39
23	RSocket Java	1.1.3
24	Simple Binary Encoding	1.27.0
25	Simplify	1.3.0
26	Spring Batch	5.0.1
27	Spring Data JPA	3.0.3
28	SpringDoc OpenAPI	2.0.2
29	TableSaw	0.43.1
30	Thymeleaf	3.1.1.RELEASE
31	Unirest Java	3.14.2
32	WireMock	3.0.0-beta.7

Fonte: (autoria própria, 2024)

O ambiente de desenvolvimento consistiu em um computador com um processador intel core i5 6400, 16 GB de RAM 2666mhz e sistema operacional Linux Mint. Não obstante, durante os testes, todos os programas aplicativos não essenciais para a execução foram fechados. Apesar disso, os testes não aconteceram em um ambiente isolado, portanto processos do sistema operacional, incluindo a interface gráfica, mantiveram-se executando, o que pode ter afetado os testes. Ambos os testes foram executados no dia 5 de dezembro de 2024.

Table 2: Tipos de assert considerados para o experimento

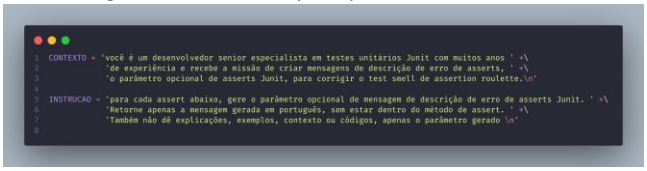
JUnit 4	JUnit 5
assertArrayEquals	assertAll
assertEquals	assertArrayEquals
assertFalse	assertEquals
assertNotEquals	assertDoesNotThrow
assertNotNull	assertFalse
assertNotSame	assertIterableEquals
assertNull	assertLinesMatch
assertSame	assertNotEquals
assertThat	assertNotNull
assertThrows	assertNotSame
assertTrue	assertInstanceOf
fail	assertNull
	assertSame
	assertThrows
	assertThrowsExactly
	assertTimeout
	assertTimeoutPreemptively
	assertTrue
	fail

Fonte: (autoria própria, 2024)

[4] https://pypi.org/project/jaro-winkler/

[5] https://platform.openai.com/docs/guides/rate-limits?context=tier-one

Figure 1: Formato de prompt utilizado nas LLMs



Fonte: (Autoria própria, 2024)

3 Resultados

A presente seção apresenta os resultados dos dois experimentos, que utilizam o mesmo código fonte python presente na Figura 2. Note que a função *acessar_arquivo* realiza as operações de geração de mensagens por LLMs e comparação de similaridade com as descrições da ferramenta geradora, lidas de um arquivo. A linha 20 e 21 realizam a geração das mensagens através de chamadas às APIs do ChatGPT e Gemini, respectivamente. Por limitação de espaço, essas chamadas omitidas.

A função main, por sua vez, realiza a chamada da função *acessar_arquivo* e aplica as técnicas de multi-thread por meio do método *ThreadPoolExecutor* da concurrent.futures, biblioteca padrão de concorrência no python [6]. Observe que a quantidade de threads utilizada é passada por parâmetro na linha 52, e corresponde ao argumento de linha de comando lido na linha 42.

As demais subseções apresentam e discutem os resultados de cada experimento.

3.1 Similaridade das Descrições de Assert

Com o intuito de verificar a razoabilidade das mensagens de descrição de assert geradas pela ferramenta, realizou-se um experimento comparativo entre elas e as geradas por Large Language Models (LLM, grandes modelos de linguagem). Os resultados de similarida estão presentes na Tabela 3.

Nota-se que, de forma geral, as mensagens geradas pela ferramenta apresentam uma similaridade alta com as mensagens geradas por LLMs, com médias gerais de 0,71 para o ChatGPT e 0,74 para o Gemini. Tais resultados podem indicar uma similaridade moderadamente boa, e que as mensagens geradas podem possuir uma coerência razoável em termos do vocabulário utilizado.

Quando observado os resultados por projeto, nota-se que o ChatGPT variou sua similaridade entre 0,58 (Logback) e 0,79 (Liquibase), enquanto o Gemini variou menos com 0,68 (Jsoup) e 0,77 (Simplify, Optaplanner, Rsocket Java, Unirest Java). Esse resultado sugere que os modelos podem ter dificuldade para gerar mensagens a depender do projeto e de seu contexto.

Não obstante, destaca-se que o Gemini apresentou similaridades ligeiramente maiores que as do ChatGPT, bem como uma variação menor entre projetos. Ainda que pequena a variação, ela demonstra as diferenças de treinamento que os modelos têm. Apesar disso, ressalta-se que ao analisar as respostas obtidas pelas LLMs, notouse que em alguns casos, as mensagens não têm relação nenhuma com o assert original, não seguem a sintaxe Java ou apresentam uma mensagem genérica como "a execução falhou". A ferramenta, por outro lado, tende a apresentar resultados constantes independentemente do projeto, tendo em vista que a lógica para criação das mensagens personalizadas é fixa e baseada na sintaxe da linguagem Java.

De forma geral, os resultados indicam que a ferramenta gera mensagens razoavelmente similares às geradas por LLMs. Portanto, pode-se concluir que os resultados gerados pela ferramenta são um bom ponto de partida ao ter em vista que os grandes modelos de linguagem foram treinados com dados reais, eles podem possivelmente representar as práticas adotadas por projetos reais e pela literatura. No entanto, conforme mencionado anteriormente, as respostas geradas pelos modelos podem não descrever corretamente os asserts a que originaram, além de possivelmente não compilar sob a sintaxe Java. Outras limitações das LLMs são o fato delas serem pagas (ao menos nos planos utilizados) e dos tempos de execução serem maiores do que o da ferramenta.

Esses resultados abrem margem para outras discussões e trabalhos futuros, dentre os quais destaca-se a realização de experimentos empíricos com usuários reais para verificar a validade das mensagens geradas pela ferramenta e comparar seu nível satisfação em relação a mensagens geradas por LLMs.

3.2 Desempenho Multithread

Os resultados das execuções com uma, quatro e oito threads podem ser observadas nas Tabelas 4, 5, e 6, respectivamente.

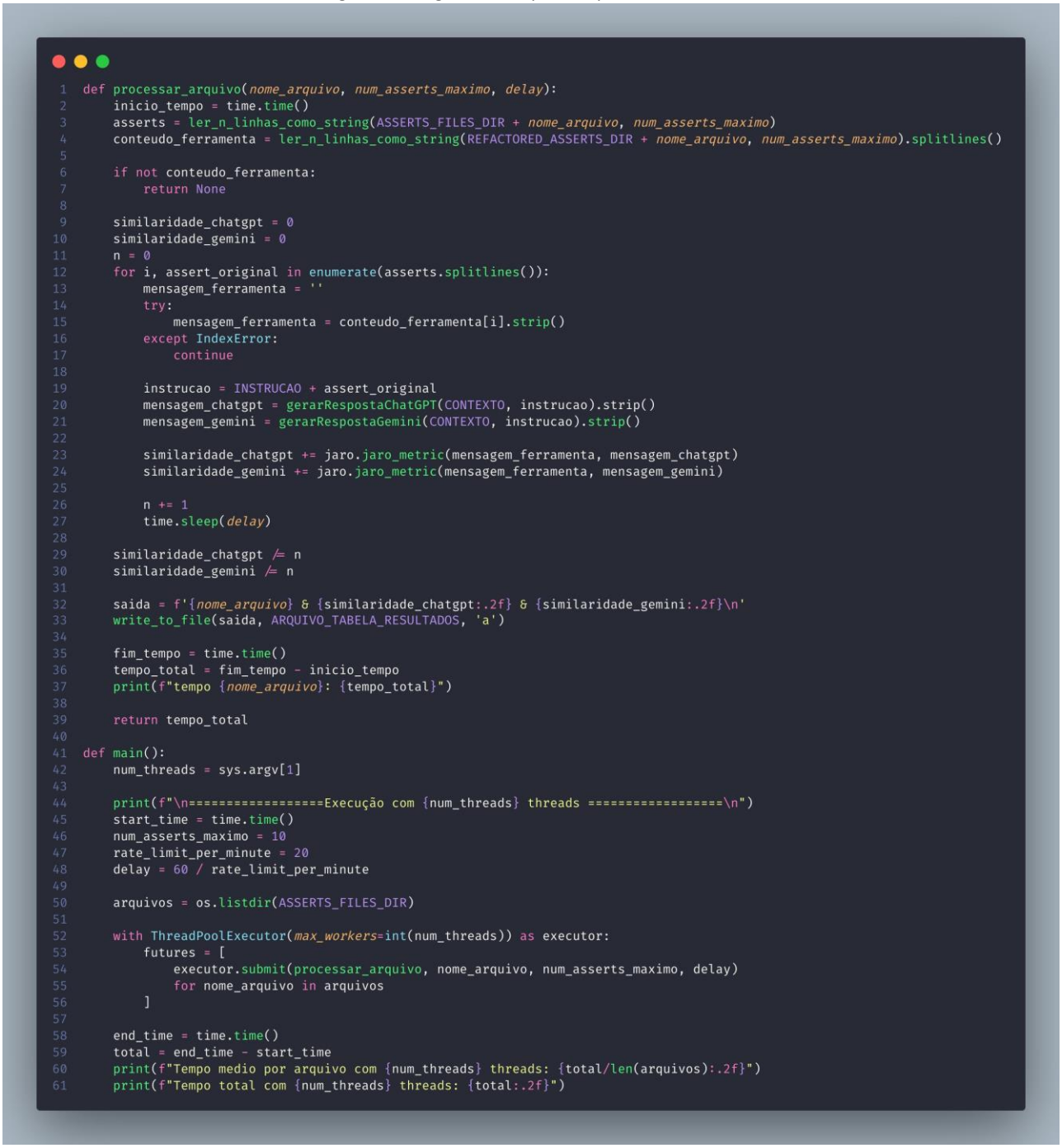
Nota-se que os resultados indicam um impacto positivo da utilização de multi-threading no desempenho do algoritmo comparativo. Quando executado em uma única thread (Tabela 4), o tempo médio de execução da comparação foi de 1379 segundos, o que equivale a 23 minutos, além de um tempo médio por arquivo de 39 segundos. Esse desempenho inicial com uma thread representa a execução sequencial do algoritmo de comparação e serve de ponto de referência para as execuções multi-thread.

Com a utilização de quatro Threads (Tabela 5), obteve-se uma redução significativa no tempo total de execução, que passou para uma média de 351 segundos, ou cerca de 6 minutos, com o tempo médio por arquivo em torno de 10 segundos. Esse resultado indica um speedup de cerca de 3,92, ou seja, uma melhoria de quase quatro vezes em relação à execução sequencial.

Ao elevar o número de threads para oito (Tabela 6), observa-se um resultado ainda melhor, com o tempo médio total caindo para 183 segundos (3 minutos) e o tempo médio por arquivo por volta de 5 segundos. Esse resultado indica uma melhora de aproximadamente 7,53 em relação a execução sequencial, indicando que o aumento do paralelismo para 8 threads continua a trazer benefícios nessas condições.

[6] https://docs.python.org/3/library/concurrent.futures.html

Figure 2: Código utilizado pelos experimentos



```python
def processar_arquivo(nome_arquivo, num_asserts_maximo, delay):
    inicio_tempo = time.time()
    asserts = ler_n_linhas_como_string(ASSERTS_FILES_DIR + nome_arquivo, num_asserts_maximo)
    conteudo_ferramenta = ler_n_linhas_como_string(REFACTORED_ASSERTS_DIR + nome_arquivo, num_asserts_maximo).splitlines()

    if not conteudo_ferramenta:
        return None

    similaridade_chatgpt = 0
    similaridade_gemini = 0
    n = 0
    for i, assert_original in enumerate(asserts.splitlines()):
        mensagem_ferramenta = ''
        try:
            mensagem_ferramenta = conteudo_ferramenta[i].strip()
        except IndexError:
            continue

        instrucao = INSTRUCAO + assert_original
        mensagem_chatgpt = gerarRespostaChatGPT(CONTEXTO, instrucao).strip()
        mensagem_gemini = gerarRespostaGemini(CONTEXTO, instrucao).strip()

        similaridade_chatgpt += jaro.jaro_metric(mensagem_ferramenta, mensagem_chatgpt)
        similaridade_gemini += jaro.jaro_metric(mensagem_ferramenta, mensagem_gemini)

        n += 1
        time.sleep(delay)

    similaridade_chatgpt /= n
    similaridade_gemini /= n

    saida = f'{nome_arquivo} & {similaridade_chatgpt:.2f} & {similaridade_gemini:.2f}\n'
    write_to_file(saida, ARQUIVO_TABELA_RESULTADOS, 'a')

    fim_tempo = time.time()
    tempo_total = fim_tempo - inicio_tempo
    print(f"tempo {nome_arquivo}: {tempo_total}")

    return tempo_total

def main():
    num_threads = sys.argv[1]

    print(f"\n=================Execução com {num_threads} threads =================\n")
    start_time = time.time()
    num_asserts_maximo = 10
    rate_limit_per_minute = 20
    delay = 60 / rate_limit_per_minute

    arquivos = os.listdir(ASSERTS_FILES_DIR)

    with ThreadPoolExecutor(max_workers=int(num_threads)) as executor:
        futures = [
            executor.submit(processar_arquivo, nome_arquivo, num_asserts_maximo, delay)
            for nome_arquivo in arquivos
        ]

    end_time = time.time()
    total = end_time - start_time
    print(f"Tempo medio por arquivo com {num_threads} threads: {total/len(arquivos):.2f}")
    print(f"Tempo total com {num_threads} threads: {total:.2f}")
```

Fonte: (Autoria própria, 2024)

Table 3: Média de similaridade de mensagens de assert geradas por LLMs em relação à ferramenta

Projeto	Média sim. Chatgpt	Média sim. Gemini
Disruptor	0.76	0.75
Logback	0.58	0.75
Cucumber JVM	0.62	0.71
Jetcache	0.74	0.70
FastJson2	0.73	0.74
Jasypt Spring Boot	0.70	0.75
Tablesaw	0.70	0.77
Liquibase	0.79	0.71
Redisson	0.72	0.73
RoaringBitmap	0.75	0.73
Unirest Java	0.76	0.77
Jsoup	0.65	0.68
Javaparser	0.71	0.71
Avro	0.72	0.71
Wiremock	0.76	0.70
Spring Data	0.72	0.72
OpenPDF	0.70	0.72
Spring Batch	0.74	0.73
Thymeleaf	0.74	0.76
Rsocket Java	0.72	0.77
Optaplanner	0.71	0.77
Jodd	0.75	0.76
Simple Binary Encoding	0.78	0.72
JDA	0.67	0.75
Dropwizard	0.74	0.75
Jfreechart	0.67	0.74
Graphhopper	0.70	0.74
Poi-tl	0.76	0.74
Ripme	0.69	0.72
Springdoc Openapi	0.73	0.70
Recaf	0.76	0.72
Simplify	0.62	0.77
Media	0.71	0.74

Fonte: (autoria própria, 2024)

Table 4: Tempos de execução com uma thread

Execução	Tempo total de execução(s)	Tempo médio por arquivo (s)
1	1381.44	39.47
2	1378.39	39.38
3	1379.65	39.42
Média	1379.82	39.42

Fonte: (autoria própria, 2024)

Table 5: Tempos de execução com quatro threads

Execução	Tempo total de execução(s)	Tempo médio por arquivo (s)
1	351.88	10.05
2	347.28	9.92
3	355.40	10.15
Média	351.52	10.04

Fonte: (autoria própria, 2024)

Table 6: Tempos de execução com oito threads

Execução	Tempo total de execução(s)	Tempo médio por arquivo (s)
1	181.92	5.20
2	178.51	5.10
3	189.10	5.40
Média	183.18	5.23

Fonte: (autoria própria, 2024)

Do ponto de vista do experimento de comparação de similaridade, a diminuição do tempo de execução obtido pelo aumento do número de threads foi satisfatório, pois representou um ganho de 86.74%, no caso da aplicação de 8 threads.

Apesar disso, o algoritmo ainda apresenta oportunidade de melhoria nos tempos de execução através da inclusão de paralelismo nas chamadas para APIs, ou seja, execução concorrente das chamadas ao ChatGPT e ao Gemini, bem como através da redução dos timeouts entre as chamadas das LLMs através do uso de requisições em lote, recurso disponível em ambas as APIs. Além disso, novos trabalhos podem verificar o limite de ganho que o aumento de threads representa para os tempos de execução, que tendem a ser limitados pela porção não paralela do código.

4 Considerações Finais

O presente artigo teve como objetivo apresentar analisar a similaridade das mensagens de assert da ferramenta de descrição de asserts e as geradas por LLM, bem como os ganhos obtidos pelo uso de multi-threading.

Quanto a análise de similaridade, os resultados indicam que a ferramenta gera mensagens cerca de 70% similares as geradas por LLMs, mas possui vantagens em relação ao primeiro por ser mais constante quanto ao formato das mensagens, sempre seguir a sintaxe java corretamente, apresentar tempos de execução menor independentemente da quantidade de asserts e por não ser pago.

Já quanto a aplicação de técnicas de multi-thread, os resultados indicam uma diminuição considerável do tempo de execução, que representa um ganho de até 7,53 vezes em relação à execução sequencial quando 8 threads são aplicadas.

Apesar disso, o presente artigo abre margem para trabalhos futuros, que incluem a realização de experimentos empíricos com pessoas reais para verificar a validade das mensagens de assert geradas pela ferramenta, a inclusão de paralelismo entre as chamadas das APIs das LLMs e o uso de chamadas em lote visando diminuir ainda mais o tempo de execução do algoritmo de

comparação e a verificação do ganho máximo que a adição de threads traria.

Referências

[1] Wajdi Aljedaani, Mohamed Wiem Mkaouer, Anthony Peruma, and Stephanie Ludi. 2023. Do the Test Smells Assertion Roulette and Eager Test Impact Students'
Troubleshooting and Debugging Capabilities?. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 29–39.

[2] Wajdi Aljedaani, Anthony Peruma, Ahmed Aljohani, Mazen Alotaibi, Mohamed Wiem Mkaouer, Ali Ouni, Christian D Newman, Abdullatif Ghallab, and Stephanie Ludi. 2021. Test smell detection tools: A systematic mapping study. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*. 170–180.

[3] Paul Ammann and Jeff Offutt. 2016. *Introduction to software testing*. Cambridge University Press.

[4] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration Code Generation via ChatGPT. arXiv:2304.07590 [cs.SE] https://arxiv.org/abs/2304.07590

[5] F GONDIM. 2006. Algoritmo de comparação de strings para integração de esquemas de dados. *Trabalho de Conclusão de Curso (Graduação)* (2006). https: //www.cin.ufpe.br/~tg/2005-2/fmg.pdf

[6] Robert C Martin. 2009. *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

[7] Railana Santana, Luana Martins, Larissa Soares, Tássio Virgínio, Adriana Cruz, Heitor Costa, and Ivan Machado. 2020. RAIDE: a tool for Assertion Roulette and Duplicate Assert identification and refactoring. 374–379. https://doi.org/10.1145/3422392.3422510

[8] Arie Van Deursen, Leon Moonen, Alex Van Den Bergh, and Gerard Kok. 2001. Refactoring test code. In *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*. Citeseer, 92–95.

[9] Zhiqiang Yuan, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, Xin Peng, and Yiling Lou. 2024. Evaluating and improving chatgpt for unit test generation. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1703–1726.