Avaliação do uso de IA generativa como auxiliar na portabilidade de scripts de testes

Title: Evaluation of the use of generative AI to assist in the portability of test scripts

Filipe Paz Reis Pinheiro[1], **Sidney de Carvalho Nogueira**[2]

[1]Departamento de computação – UFRPE
Rua Dom Manoel de Medeiros, s/n, Dois Irmãos. Recife - PE - Brasil

[2]Departamento de computação – UFRPE
Rua Dom Manoel de Medeiros, s/n, Dois Irmãos. Recife - PE - Brasil

filipe.paz@ufrpe.br, sidney.nogueira@ufrpe.br

Abstract. *Test automation frameworks are becoming more advanced and complementary. This work investigates the potential of ChatGPT v3.5 to perform the migration of test scripts between Selenium and Cypress, evaluating the accuracy and reliability of the generated codes. After selecting scripts and creating prompts, the conversion success rate was measured, verifying whether the converted scripts executed correctly without syntax errors and with the same result as the original. A satisfactory conversion rate was found, 100% from Selenium to Cypress and 97.73% from Cypress to Selenium. However, it highlighted the need for manual adjustments to ensure the success of the migration, since there were converted scripts with incorrect results.*

Keywords. *Code conversion; ChatGPT; Artificial Intelligence; LLM; Selenium; Cypress; Automated test.*

Resumo. *Os frameworks de automação de testes estão se tornando mais avançados e complementares. Este trabalho investiga o potencial do ChatGPT v3.5 para realizar a migração de scripts de testes entre Selenium e Cypress, avaliando a precisão e a confiabilidade dos códigos gerados. Após selecionar scripts e elaborar prompts, foi mensurada a taxa de sucesso da conversão, verificando se os scripts convertidos executaram corretamente sem erros de sintaxe e com mesmo resultado do original. Constatou uma taxa de conversão satisfatória, 100% de Selenium para Cypress e 97.73% de Cypress para Selenium. Contudo, evidenciou a necessidade de ajustes manuais para garantir o sucesso da migração, já que ocorreu scripts convertidos com resultados errados.*

Palavras-Chave. *Conversão de código; ChatGPT; Inteligência Artificial; LLM; Selenium; Cypress; Testes automatizados.*

1. Introdução

Historicamente, qualidade de *software*, é um tema que vem sendo discutido desde décadas passadas. Em 1979, já era abordado o processo de validar e verificar o que estava sendo desenvolvido [Axelsson and Skoglund 2016]. Atualmente, o assunto continua relevante, à medida que os *softwares* estão ficando cada vez mais complexos e presentes no cotidiano das pessoas [Lagstedt et al. 2021], cresce a necessidade de ter garantia de qualidade robusta e eficiente nesses programas, tornando, assim, a prática de testar uma atividade essencial no ciclo de desenvolvimento, sendo essa, a prática mais difundida para se obter qualidade na produção das *features* [Garousi et al. 2020, Siva et al. 2023].

O processo de testes consome de 30 a 60% do custo do ciclo de vida [Polo et al. 2013], podendo ser realizado de duas formas: manualmente ou automaticamente. Testes manuais de sistemas são feitos de forma similar a um usuário final utilizando a interface do sistema, entretanto, manter apenas testes manuais é muito custoso, pois, à medida que o programa cresce em tamanho e complexidade, o tempo para serem executados cresce junto, tornando praticamente inviável. Como resultado, é necessário executar apenas uma parte dos testes, diminuindo a cobertura e a confiança do produto sendo lançado [Garousi et al. 2018, Leotta et al. 2023]. Por conseguinte, testes automatizados ajudam a otimizar esforço, já que são realizados com o suporte de ferramentas que executam automaticamente *scripts* [Garousi et al. 2018], frequentemente e rapidamente [Leotta et al. 2023], reduzindo custos, aumentando a qualidade e diminuindo chance de erro humano [Polo et al. 2013, Wiklund et al. 2017].

A ferramenta utilizada para automatizar os testes é um *software* específico, que não é o *software* que está sendo testado. Chamados, também, de *frameworks*, foram surgindo à medida que a área de qualidade se desenvolvia, para controlar a execução de testes e fazer a comparação dos resultados reais com os resultados esperados [Huizinga and Kolawa 2007]. Todavia, é necessário esforço para a escrita de *scripts*, visto que essa atividade é feita utilizando alguma linguagem de programação e, no caso de automação para sistemas *web*, com adição de uma biblioteca para lidar com a execução no navegador [Leotta et al. 2023].

Frameworks de automação são essenciais para a automação dos testes; evoluem e se complementam nas funções, de forma que às vezes pode ser necessário reimplementar *scripts* feitos em um *framework* na linguagem de outro *framework*. Selenium e Cypress são duas das mais populares ferramentas de testes automatizados, utilizadas por profissionais de qualidade de *software*, Selenium, teve seu surgimento em 2004 com uma proposta de ser suportado para várias linguagens [Software Freedom Conservancy 2024], e Cypress, criado 10 anos depois, em 2014, com o intuito de ser fácil, rápido e confiável [Cypress.io 2024, Kinsbruner and Bahmutov 2022]. Em agosto de 2023, Selenium possuía cerca de 180.000 usuários no repositório do GitHub, enquanto Cypress, 699000 [GH and Badkar 2023], contudo, esse dado em julho de 2024 já é cerca 273000 usuários para Selenium e 1344000 para Cypress [GitHub 2024a, GitHub 2024b], mostrando a crescente de popularidade, sobretudo do Cypress. Na Figura 1 abaixo, são mostradas algumas ferramentes para testes automatizados e sua respectiva quantidade aproximada de usuários no GitHub em julho de 2024 [GitHub 2024a, GitHub 2024b,

GitHub 2024c, GitHub 2024d, GitHub 2024e, GitHub 2024f, GitHub 2024g].

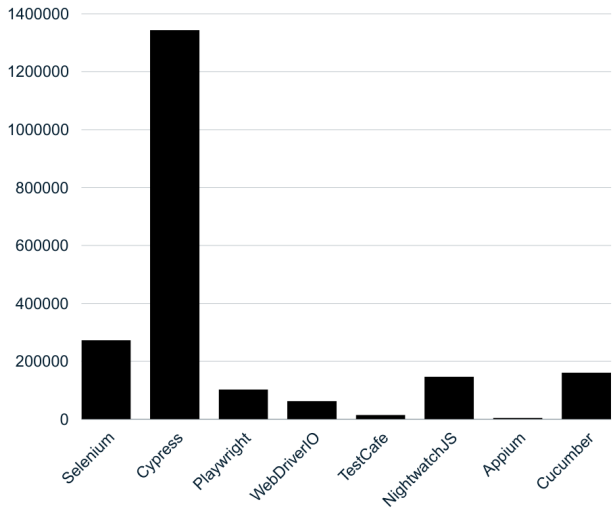

Figura 1. Gráfico mostrando os usuários das ferramentas de testes automatizados no GitHub em julho de 2024

Como o custo de realizar automação é significante, os *scripts* precisam ser executados muitas vezes, de modo a ter um retorno do investimento [Wiklund et al. 2017, Wang 2018]. Se houver a necessidade de portar (migrar) os *scripts* de uma tecnologia para outra, esse custo pode crescer ainda mais, já que será necessário reescrever os *scripts* para serem executados na nova ferramenta, alocando mais tempo e recursos para essa atividade [Harrell et al. 2018].

Os profissionais associados com o processo de automação de testes podem sentir a necessidade de migrar os *scripts*. Se no projeto surgir a necessidade de ter uma maior compatibilidade com tecnologias existentes e ter testes executando em paralelo, utilizar Selenium é mais indicado, isso devido a esse *framework* suportar diversas linguagens de programação e permitir testes em mais navegadores que Cypress. Porém, se houver necessidade de testes mais rápidos, seria interessante ter *scripts* em Cypress, isso devido a sua execução direto no navegador. Outro fator relevante é a preferência da equipe, em casos de mudanças, os novos integrantes podem se deparar com códigos no *framework* em que não têm tanta familiaridade, decidindo pela migração.

Inteligência artificial (IA) vem sendo amplamente usada, tendo sido utilizada na campanha presidencial da Argentina [G1 2023], sobretudo a generativa para aumentar as habilidades de engenharia [Sogeti 2023], constatando que 84% das organizações dos Estados Unidos estão usando esse tipo de IA para otimizar seus processos de testes [Sogeti 2023]. Em outra pesquisa, constatou que a IA generativa possui uma adoção recorde no mercado, cerca de 2 a cada 3 usuários (correspondendo a 67% dos pesquisados) reportaram que suas empresas estão usando essas IAs no ambiente de trabalho [O'Reilly Media 2023]. Seu uso pode ser diverso, desde atividades simples, como pedir receitas, até as mais complexas, como cálculos matemáticos. Contudo, seu uso mais comum é feito por profissionais de tecnologia nas suas atividades: 77% dos en-

trevistados afirmaram que usam para auxiliar na programação, enquanto 70% usam na análise de dados [O'Reilly Media 2023].

Um problema encontrado no uso de IA generativa é que nem sempre os resultados são corretos e precisos, em um estudo realizado na universidade Purdue foi constatado que o ChatGPT produziu mais de 50% de respostas incorretas [Kabir et al. 2023]. Isto posto, é relevante a indagação sobre a taxa de sucesso para a tradução de um *script* escrito em um uma linguagem para outra.

Entretanto, não há muitas publicações estritamente relacionadas ao uso de IA generativa como auxiliar na portabilidade de *scripts* de testes, por exemplo, de Selenium para Cypress ou vice-versa. Porém, em 2016, há à publicação de um artigo em que foi proposto uma solução para converter códigos entre as plataformas *mobiles*. Partindo para o âmbito da IA generativa, mais especificamente o ChatGPT, é possível encontrar trabalhos demonstrado sua aplicabilidade, como feito para mostrar sua capacidade de prever a legitimidade de notícias [Caramancion 2023]. Em outro trabalho, foi estudada a eficácia da autoavaliação de trabalhos de alunos utilizando o ChatGPT v3.5 e v4, com objetivo de analisar a precisão e confiabilidade dessa IA generativa [Altamimi 2023]. Sobre tradução de códigos, em um artigo foi feita a comparação do resultado obtido utilizando um modelo apresentado por eles e utilizando o ChatGPT [Mahmoud et al. 2023].

Portanto, como mencionado anteriormente, não há muitas publicações recentes sobre o tema específico proposto, embora haja o estudo do uso para outros propósitos e demonstrações de sua eficácia. Desse modo, esta pesquisa tem como principal contribuição a análise do uso de IA generativa, mais especificamente do *chatbot* ChatGPT v3.5, com intuito de fazer a portabilidade de *scripts*, de Selenium para Cypress e vice-versa, avaliando se os *scripts* obtidos pela tradução produzem o resultado esperado, sem erros sintáticos ou semânticos e quantificando a taxa de sucesso e erro dessa operação. Portanto, responderá as seguintes perguntas: é possível usar uma IA generativa com o intuito de transformar *scripts* escritos em um *framework* para outro? Mais especificamente, qual a taxa de sucesso nessa portabilidade?

A estrutura desse trabalho está organizada da seguinte forma. Na próxima seção são introduzidos conceitos importantes para o entendimento do trabalho. A seção 3 apresenta trabalhos relacionados. A Seção 4, descreve a metodologia no trabalho, a Seção 5 discute os resultados obtidos. Conclusão e trabalhos futuros são discutidos na Seção 6.

2. Fundamentação Teórica

2.1. Qualidade de *Software* e testes

Uma definição de qualidade de *software* é o sistema atender as necessidades e ter os requisitos averiguados pelas partes interessadas [Axelsson and Skoglund 2016]. Contudo, essa definição pode mudar conforme a perspectiva de cada pessoa envolvida com o *software*, por exemplo, é diferente para os clientes e para os desenvolvedores. Desse modo, há vários fatores que definem a qualidade: ter poucos defeitos de requisitos, baixo potencial de defeitos, uso de código reutilizável certificado, baixas taxas de defeitos de gravidade 1 e 2 [Jones and Bonsignour 2011]. Tabela 1 mostra os 12 principais fatores de qualidade de software.

Tabela 1. Os 12 fatores de qualidade de *software* mais eficazes [Jones and Bonsignour 2011]

1.	Baixos defeitos em potencial
2.	Método efetivo de prevenção de defeito
3.	Alta eficiência de detecção de defeitos
4.	Alta eficiência de remoção de defeitos
5.	Uso de inspeções pré-teste
6.	Uso de análise estática de pré-teste
7.	Uso de design formal de caso de teste
8.	Boa facilidade de aprendizado
9.	Boa facilidade de uso
10.	Bom suporte técnico
11.	Alta satisfação do usuário
12.	Boa garantia

Todavia, a prática para ter qualidade de *software* mais difundida é testar, sendo essa uma atividade essencial no ciclo de desenvolvimento do *software* [Garousi et al. 2020, Siva et al. 2023]. Esses testes podem ser feitos majoritariamente de duas formas: testes manuais e testes automatizados. Teste manual é feito com um testador, onde ele executa passos a fim de realizar a verificação de determinada função ou requisito do sistema, comparando se o resulto obtido é, também, o esperado, ou seja, o *tester* age como se fosse um usuário final [Ramya et al. 2017]. Já o teste automatizado, é feito com ajuda de um *software*, onde são escritos *scripts* para cobrir algum cenário de teste do sistema. Quando esses *scripts* são executados, o computador segue uma sequência de passos, obtendo resultados mais rápidos em comparação com o manual, sendo uma popular alternativa para reduzir o custo do teste de *software* [Tahvili et al. 2018].

2.2. Selenium e Cypress

Selenium é um *framework* de testes com código aberto, focado em aplicações *web*. Pode ser usado nos principais navegadores do mercado, com suporte a várias linguagens de programação, múltiplas janelas e abas, características que seu principal concorrente, o Cypress, não possui [Ramya et al. 2017, Kinsbruner and Bahmutov 2022]. As Figuras 2 e 3 mostram o mesmo *script* escrito para o Selenium nas linguagens Pyhton e Java, respectivamente.

Cypress é outro *framework* focado em aplicações *web*. É executado diretamente do navegador e seus códigos são escritos com a linguagem JavaScript. Além disso, tem uma arquitetura única que possibilita o controle tanto do *front* quanto do *backend* da aplicação [Pelivani et al. 2022]. Devido a sua arquitetura, em comparação com o Selenium, a execução dos testes é mais rápida, o processo de *debugging* mais fácil, com mais confiabilidade e menos testes inconsistentes, isto é, testes que têm resultados diferentes em execuções distintas, possuindo um filtro para essas ocorrências [Kinsbruner and Bahmutov 2022].

```python
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()

driver.get("https://www.selenium.dev/selenium/web/web-form.html")

title = driver.title

driver.implicitly_wait(0.5)

text_box = driver.find_element(by=By.NAME, value="my-text")
submit_button = driver.find_element(by=By.CSS_SELECTOR, value="button")

text_box.send_keys("Selenium")
submit_button.click()

message = driver.find_element(by=By.ID, value="message")
text = message.text

driver.quit()
```

Figura 2. Código escrito em Python para o Selenium

2.3. SonarQube

SonarQube [SonarSource 2024] é uma ferramenta de análise de código projetada para detectar problemas e fornecer dados sobre a qualidade da codificação. Após ser utilizado, algumas das métricas fornecidas são:

- Quantidade de linhas de códigos;
- Quantidade de linhas de códigos duplicados;
- *Code Smells*
- Débito técnico;
- Complexidade.

A métrica de quantidade de linhas de códigos retorna o total de linhas que contêm pelo menos um caractere, que não é um espaço em branco, nem uma tabulação, nem parte de um comentário. Quantidade de linhas de códigos duplicados é o número de linhas envolvidas em duplicações. *Code Smells* é o número total de problemas que impactam a manutenibilidade. Débito técnicos consiste em uma medida (em minutos) sobre o esforço necessário para consertar todos os problemas de manutenibilidade. Complexidade é dividida em dois tipo: complexidade ciclomática e complexidade cognitiva. A complexidade ciclomática é uma métrica usada para calcular o número de caminhos através do código. A complexidade cognitiva é uma quantificação de quão difícil é entender o fluxo de controle do código. Mais detalhes dessas métricas podem ser vistas na documentação oficial do SonarQube[1].

[1] https://docs.sonarsource.com/sonarqube/latest/user-guide/code-metrics/metrics-definition/

```java
package dev.selenium.getting_started;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

import java.time.Duration;

public class FirstScript {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();

        driver.get("https://www.selenium.dev/selenium/web/web-form.html");

        driver.getTitle();

        driver.manage().timeouts().implicitlyWait(Duration.ofMillis(500));

        WebElement textBox = driver.findElement(By.name("my-text"));
        WebElement submitButton = driver.findElement(By.cssSelector("button"));

        textBox.sendKeys("Selenium");
        submitButton.click();

        WebElement message = driver.findElement(By.id("message"));
        message.getText();

        driver.quit();
    }
}
```

Figura 3. Código escrito em Java para o Selenium

2.4. Inteligência Artificial (IA)

Inteligencia artificial é um termo complexo, contudo, uma das várias definições amplamente aceitas, é o estudo de ensinar o computador a fazer coisas que as pessoas, no geral, fazem melhor [Rich et al. 2009, Russell and Norvig 2010]. Comumentemente, a IA é dividida em algumas disciplinas. Processamento de linguagem natural, consiste no computador ser capaz de entender a língua humana. Representação de conhecimento e raciocínio, que é sobre a capacidade de armazenar o que se conhece para usar esse conhecimento para resolver problemas existentes. Aprendizado de máquina, adaptação de novas circunstâncias e extrapolação de dados iniciais. Visão computacional, sendo o estudo do computador ser capaz de perceber objetos. Robótica, possibilidade de manipular objetos e mover [Russell and Norvig 2010].

Afunilando, é interessante discorrer sobre o que é IA generativa: uma subdivisão de aprendizado de máquina, consistindo em treinar um modelo com dados específicos para que ele possa resultar em um novo dado semelhante com o que lhe foi treinado, que pode ser usado em diversos âmbitos (texto, imagem, áudio, vídeo). Como exemplo, imagine o caso de termos dados de cavalos, podemos então usar esses dados para treinar um modelo, de modo que ele gerará cavalos que não existam nos dados originais, como

mostrado na figura 4 [Foster 2022, Oh and Shon 2023]. Logo, podemos concluir que é possível utilizar uma IA generativa para fazer tradução de códigos entre linguagens de programação, assim como tradução de *scripts* de um *framework* para outro.

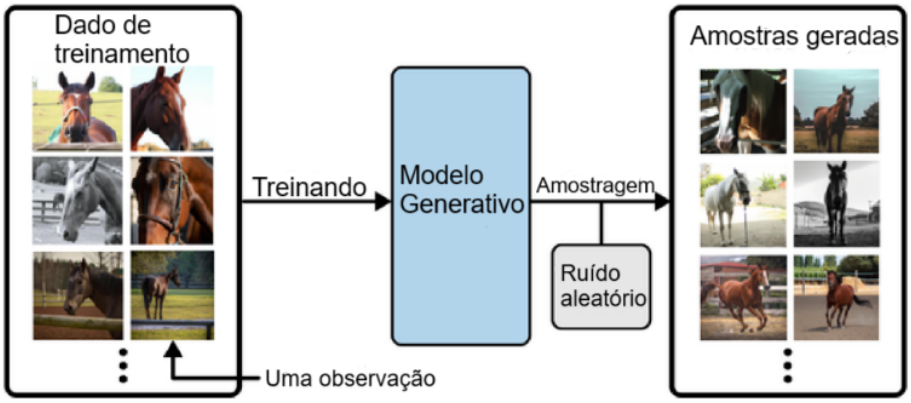

Figura 4. Modelo generativo treinado para gerar imagens realística de cavalos [Foster 2022]

2.4.1. ChatGPT

ChatGPT é um multi-funcional *bot* de texto que surgiu em 2022. Ele se baseia em conversas humanas por meio de textos inseridos no seu sistema, comumente chamados de *prompt*. Com a ajuda de uma enorme coleção de dados, o ChatGPT é capaz de fornecer respostas em uma gama de áreas, por exemplo, ajudar na resolução de problemas de um exame, com também na produção de um trabalho acadêmico [Ahmed and Sharo 2023]. A Figura 5 ilustra o uso de ChatGPT para realizar a conversão de um código em Python para Javascript.

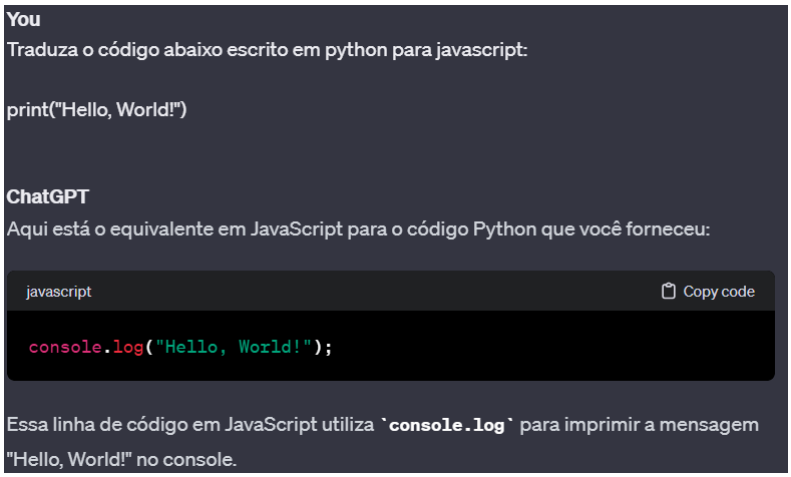

Figura 5. ChatGPT v3.5 traduzindo um código simples de Python para Javascript

3. Trabalhos relacionados

Apesar de temas relacionados a IA estar popular, o uso dela para migrar *scripts* não foi algo tão estudado ainda, visto que é algo bastante recente na ciência. Por outro lado, o estudo de conversão de código é amplamente realizado. O trabalho de [El-Kassas et al. 2016] é um exemplo de pesquisa sobre conversão de código, especificamente, nesse caso, entre as plataformas mobile Android, utilizando Java 8, e Windows Phone 8, utilizando C#. Neste trabalho, os autores usam uma abordagem baseada em *Extensible Stylesheet Language Transformation* (XSLT) e expressões regulares para facilitar a conversão. Como resultado, foi obtido números bem consistentes em comparação com outras ferramentes já existentes no mercado. Segundo os autores, o *Integrated Cross-Platform Mobile Applications Development* (ICPMD), solução proposta por eles, foi capaz de melhorar a velocidade, uso de memória e tamanho da aplicação gerada por essa ferramenta. A Figura 6 abaixo mostra o resultado dessa solução quando aplicada a um código escrito em Java 8, para um código traduzido para o C#.

Input Java Source Code (Android Platform)	Intermediate XML Representation	Output C# Source Code (WP8 Platform)

…
public class WebViewActivity extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
WebView wv = (WebView) findViewById(R.id.webview1);
WebSettings webSettings = wv.getSettings();
webSettings.setBuiltInZoomControls(true);
 	
wv.loadUrl("file:///android_asset/Index.html");
}
}
…
	…
	<comment type="singleline" source="original">
	public class WebViewActivity extends Activity </comment>
	<pageclass type="suggested" source="ICPMD">
	<param name="name" value="WebViewActivity" />
	<param name="parent" value="" />
	<param name="projectname" value="WebView" />
	</pageclass>
	<block_start block="pageclass"/>
	<comment type="multiline" source="original comment"> /** Called when the activity is first created. */</comment>
	<comment type="singleline" source="original">
	@Override public void onCreate(Bundle savedInstanceState)
	</comment>
	<constructor type="suggested" source="ICPMD"> …
	<comment type="singleline" source="Composite pattern step 1 out of total steps 1">
	wv.loadUrl("file:///android_asset/Index.html");
	</comment>
	<pattern category="composite" name="Webview" type="suggested" source="ICPMD">
	<wp>wv.Navigate(new Uri("android_asset/Index.html", UriKind.RelativeOrAbsolute));</wp></pattern>
	<block_end block="constructor"/> …
	…
		…
		public partial class WebViewActivity : PhoneApplicationPage
		{
		//original comment: /** Called when the activity is first created. */
		//original: @Override public void onCreate(Bundle savedInstanceState)
		public WebViewActivity()
		{
		InitializeComponent();
		//suggested comment : super.onCreate(savedInstanceState);
		//suggested comment : setContentView(R.layout.main);
		//suggested comment : WebView wv = (WebView) findViewById(R.id.webview1);
		//suggested comment : WebSettings webSettings = wv.getSettings(); //suggested comment : webSettings.setBuiltInZoomControls(true);
		//Composite pattern step 1 out of total steps 1:
		wv.loadUrl("file:///android_asset/Index.html");
		wv.Navigate(new Uri("android_asset/Index.html", UriKind.RelativeOrAbsolute));
		}
		}
		…

Figura 6. Conversão de código de Java para C# [El-Kassas et al. 2016]

Sobre o ChatGPT, é possível encontrar trabalhos demonstrado sua aplicabilidade, como feito para mostrar sua capacidade de prever a legitimidade de notícias com uma precisão de 100% em um tempo de apenas 1 segundo [Caramancion 2023]. Já no trabalho publicado por [Altamimi 2023], foi estudada a eficácia da autoavaliação de trabalhos de alunos utilizando o ChatGPT v3.5 e v4, com objetivo de analisar a precisão e confiabilidade dessa IA generativa, avaliando diversas áreas acadêmicas e concluindo que o ChatGPT demonstra ser eficiente no processo de avaliação.

No trabalho feito por [Mahmoud et al. 2023], já é encontrado, também, resultados de conversão usando IA generativa. Nesta pesquisa, o estudo é baseado na migração de códigos em Swift, do iOS, para java, do Android, onde é proposto uma ferramenta própria dos autores para concluir o objetivo de conversão. Em números, a ferramenta da pesquisa conseguiu obter uma acurácia geral de 89.3%. Porém, foi também utilizado o

ChatGPT 3.5 para comparar os resultados e surpreendentemente, ao descreverem a biblioteca correta para a IA generativa usar, foi obtido uma acurácia de 100%, superando a ferramente dos autores.

Por fim, no trabalho de [Yu et al. 2023], temos uma pesquisa voltado a responder 3 perguntas:

1. Qual é o desempenho de uma *Large Language Model* (LLM) em gerar *scripts* de teste baseando na descrição via linguagem natural em cenários de aplicativos específicos?
2. Qual é o desempenho de uma LLM em migrar *scripts* de teste entre plataformas, por exemplo entre *Android* e *iOs* diferentes do mesmo aplicativo?
3. Qual é o desempenho de uma LLM em migrar *scripts* de teste entre aplicativos diferentes, mas que têm funcionalidades parecidas?

Para a Pergunta 1, o foco foi apenas em 2 cenários, onde, o ChatGPT foi capaz de gerar *scripts* para ambos, porém, apenas um teve resultado satisfatório, enquanto o outro foi gerado com diversos problemas, falhando em ter algum código relevante para o respectivo cenário. Já na Pergunta 2, foi concluído que o código migrado mostrava uma impressionante precisão, sendo confiável, principalmente quando são passadas uma gama de informações consideradas necessárias para o sucesso do objetivo de conversão, mas, ainda assim, não descartando a necessidade do trabalho humano de revisar e fazer algumas correções. Por último, na Pergunta 3, o ChatGPT também demostrou certa eficiência para conclusão dessa atividade, contudo, a necessidade de um testador humano não é descartada, pois, o resultado obtido pela LLM, depende diretamente de como e quais informações são inseridas nele.

Portanto, como visto, esse é um tema que estar em ascensão e é bastante relevante no meio científico. Além disso, diferentemente dos trabalhos citados acima, essa pesquisa estudará a eficácia em usar o ChatGPT para converter *scripts* entre os dois principais *frameworks* de teste existentes para automação de interface web.

4. Metodologia

Este trabalho realiza uma avaliação sobre a efetividade no uso de ChatGPT para conversão entre os *frameworks* Cypress e Selenium. Nessa avaliação, são consideradas as seguintes métricas:

- taxa de testes convertidos com sucesso;
- taxa de testes convertidos que reproduzem o mesmo resultado;
- taxa de testes convertidos com erro de sintaxe;
- taxa de testes convertidos que não reproduzem o mesmo resultado.

Para calcular a primeira taxa, é utilizado a seguinte fórmula:

$$\frac{X}{Y} * 100$$

Onde o X equivale ao total de testes convertidos e o Y equivale ao total de testes.

As outras três taxas seguem fórmulas parecidas, sendo o da segunda:

$$\frac{S}{X} * 100$$

Onde o S equivale ao total de testes convertidos que executaram sem erro e têm o mesmo resultado do *script* original. O X tem o mesmo significado do anterior.

Já o da terceira taxa, a fórmula é a seguinte:

$$\frac{Z}{X} * 100$$

Onde o Z equivale ao total de testes convertidos que apresentaram erro de sintaxe e o X o mesmo significado da primeira taxa.

E o da quarta taxa:

$$\frac{F}{X} * 100$$

Onde o F equivale ao total de testes convertidos que não reproduziram o mesmo resultado obtido com o teste original e o X o mesmo significado dos anteriores. É importante ressaltar que, nas bases originais, cada arquivo pode ter vários testes, pois estão em uma suite, por exemplo no caso do ”Describe”do Cypress. Então, cada ”def”existente nos códigos de Selenium é considerado um único teste, como também cada ”it”do Cypress é considerado um único teste.

O primeiro passo da avaliação foi selecionar duas bases de *scripts*, uma para fazer a conversão de Selenium para Cypress e a outra para fazer o inverso. Essas bases podem ser encontradas acessando o repositório oficial dos *frameworks*. Para o Selenium, foi utilizado alguns *scripts* obtidos no repositório do Selenium. Para Cypress, foram obtidos alguns no repositório do Cypress. Essas bases foram escolhidas, pois são *scripts* oficiais de ambos os *frameworks* e podem servir como exemplos e tutoriais para ajudar na curva de aprendizado de quem deseja aprender e aplicar seus benefícios em projetos *web*. Em seguida, foi feita uma pré-conversão do *script* e após isso, é utilizado o ChatGPT para ele fazer a conversão de um *script* em um dos *frameworks* para o outro. Por fim, o código convertido é executado, analisado, segunda as métricas citadas anteriormente. Esse processo está descrito no diagrama da Figura 7.

4.1. Pré-conversão do *script*

Nesta etapa, o código obtido pela base de *script* é analisado e, se for o caso, editado. Por exemplo, na base referente ao Cypress, os códigos possuem muitos comentários. No geral, comentários em códigos são bem vistos, já que ajudam na compreensão e manutenibilidade. Porém, em alguns casos, como no repositório do Cypress, existirem comentários muito extensos, que são úteis para iniciantes, torna-o um código de baixa qualidade e de

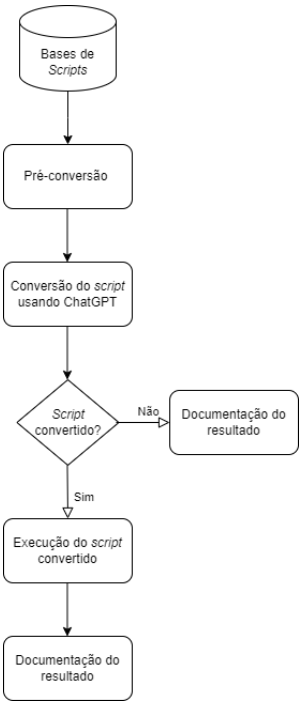

Figura 7. Diagrama geral do processo de conversão de *scripts*

difícil compreensão, ocasionando os *code smells* [Jabrayilzade et al. 2021]. Por este motivo, os comentários do código escrito em Cypress foram manualmente removidos, de modo que não prejudiquem o processo de conversão pela IA.

Quanto a base referente ao Selenium, nota-se que está faltando a chamada da função para execução do teste. Portanto, foi adicionado manualmente no final do *script* essas chamadas, não alterando em nada o conteúdo original.

Os exatos *scripts* utilizados na conversão podem ser encontrados no repositório do GitHub. Para os códigos originalmente em Cypress, acessando este link. Paras os em Selenium, acessando este outro link.

4.2. Conversão do *script*

Para realizar a conversão foi necessário elaborar um *prompt* solicitando ao ChatGPT a realização da conversão desejada. Este *prompt* foi baseado no *prompt* utilizado pelos autores em [Yu et al. 2023]. A primeira parte consiste na ambientação, onde é informado o contexto, *framework* utilizado no *script* original e o que queremos que seja gerado como resultado da tradução. A segunda parte, é o *script* original o qual desejamos que seja convertido. O modelo geral, utilizado em ambos os casos de conversão, Selenium para Cypress, e de Cypress para Selenium, podem ser vistos a seguir. Para o caso de Selenium para Cypress foi utilizado o seguinte *prompt*:

```
You are a software testing engineer, and you run a web
application on a Chrome browser. You are asked to do a test
script migration from Selenium using Python to Cypress.
Here is the Selenium code:
```

```
<Código em Selenium>
```

Para o caso de Cypress para Selenium foi utilizado o seguinte *prompt*:

```
You are a software testing engineer, and you run a web
application on a Chrome browser. You are asked to do a test
script migration from Cypress to Selenium with Python.
Here is the Cypress code:

<Código em Cypress>
```

A partir do *prompt* preparado, foi utilizada a técnica *single shot*, isto é, toda a informação é enviada para o LLM em uma única interação com o modelo[Ige et al. 2024]. O LLM utilizado para esse procedimento foi o GPT 3.5 Turbo através de interface da ferramenta ChatGPT. Na ferramenta foi utilizado um novo *chat* para cada teste a ser convertido, portanto, o modelo só usou como entrada um único teste por vez, não armazenando o contexto das conversões anteriores.

4.3. Documentação do resultado em caso da conversão

Há situações em que o ChatGPT pode não ser capaz de realizar a tarefa solicitada. Isso acontece devido a diversos fatores, desde incapacidade técnica do modelo de linguagem ou por questões éticas. Para o processo conversão de *scripts*, não é diferente. Portanto, quando a migração de código para outro *framework* não for possível, o resultado também é analisado, identificando e documentando as razões da falha. Para essa documentação, foi criado um arquivo que contém todas as respostas detalhadas fornecidas pelo ChatGPT, incluindo explicações sobre o que foi realizado e o novo código sugerido. Dessa forma, é possível obter métricas que indicam o sucesso ou insucesso do processo solicitado.

Nos casos em que o ChatGPT retorna um *script* convertido para o *framework* desejado, o resultado também é documentado nos moldes do caso de não conversão: criando um arquivo com a resposta completa da IA.

4.4. Execução do *script* convertido

A conversão bem sucedida do *script* torna possível analisar a eficácia do processo de tradução. Isto é feito executando o código obtido em uma máquina contendo *Windows* 11, analisando se o código executa sem erros de sintaxe e com os mesmos resultados retornados com a execução do código no *framework* original. Com isso, algumas métricas importantes são capturadas, de modo que seja possível avaliar a taxa de sucesso e os motivos de possíveis falhas.

Ainda nesta etapa, é verificado se o resultado de ambos os *scripts*, original e o convertido, são congruentes. Em casos afirmativos de falha, tanto ao nível de execução quanto de conversão, há investigação e documentação do motivo, esclarecendo se o contorno da falha é de fácil resolução.

5. Resultados

Os resultados da avaliação descrita acima são apresentados nesta seção. Para execução, foi escolhido arbitrariamente 17 *scripts* dos repositórios oficiais dos *frameworks* Sele-

nium e Cypress. Como Selenium é uma ferramenta que pode ser escrita usando diversas linguagens, foi optado por pegar os códigos usando Python por questões de familiaridade e popularidade da linguagem[Tiobe 2025]. Os exatos *scripts* após realizar a etapa de pré-conversão podem ser encontrados nos links dos repositórios do GitHub, para Cypress, este link, para Selenium, este outro link .

5.1. Resultados do processo de conversão

Neste tópico, será discutido os resultados do processo de conversão, isto é, quantidade de testes que foram convertidos e que não foram, quantos foram executados com sucesso, obtendo o mesmo resultado do *script* original, quantos foram executados, mas que obtiveram um resultado diferente em comparação com o original, por exemplo, no original o teste passa e no convertido o teste falha, e quantos não executaram, indicando algum erro de sintaxe, por exemplo. Sobre os testes, para os dados aqui postos, cada validação do arquivo é considerado um teste diferente, ou seja, um único arquivo do *framework* pode ter vários testes diferentes, além de que o repositório do Cypress contém mais testes em comparação com o do Selenium, portanto, sendo o total de 32 testes selecionados para o caso de Selenium para Cypress e 44 testes no caso de Cypress para Selenium. Nos gráficos das Figuras 8, 9, 10 e 11 é possível observar as principais sínteses desses resultados, onde ”Sucesso”são testes que executaram e obtiveram o mesmo resultado do original e ”Falha”são testes que executaram e obtiveram resultado diferente do original.

Figura 8. Dados da conversão de Selenium para Cypress

A partir destes gráficos, conseguimos extrair dados relevantes. Com a tradução do Selenium com Python para o Cypress, o ChatGPT realizou a conversão de todos os 32 testes, ou seja, 100% dos casos de testes foram migrados com sucesso. Contudo, um teste ter sido migrado não significa que o mesmo foi convertido corretamente. Desses

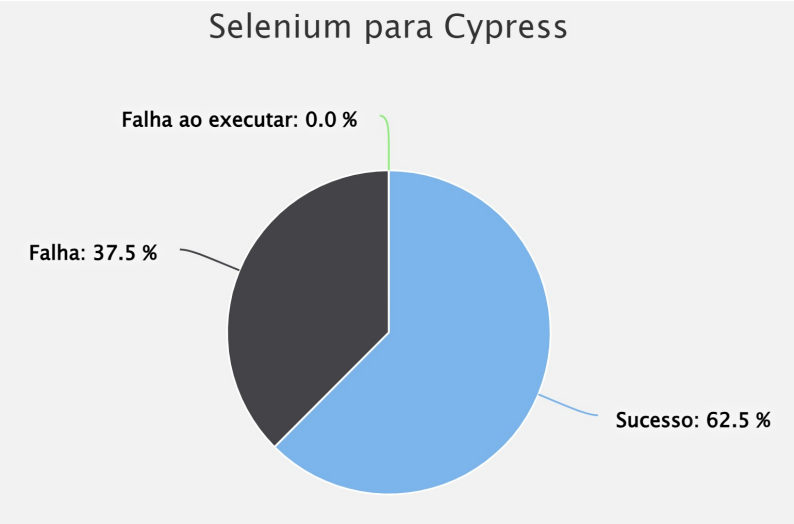

Figura 9. Dados da execução de Selenium para Cypress

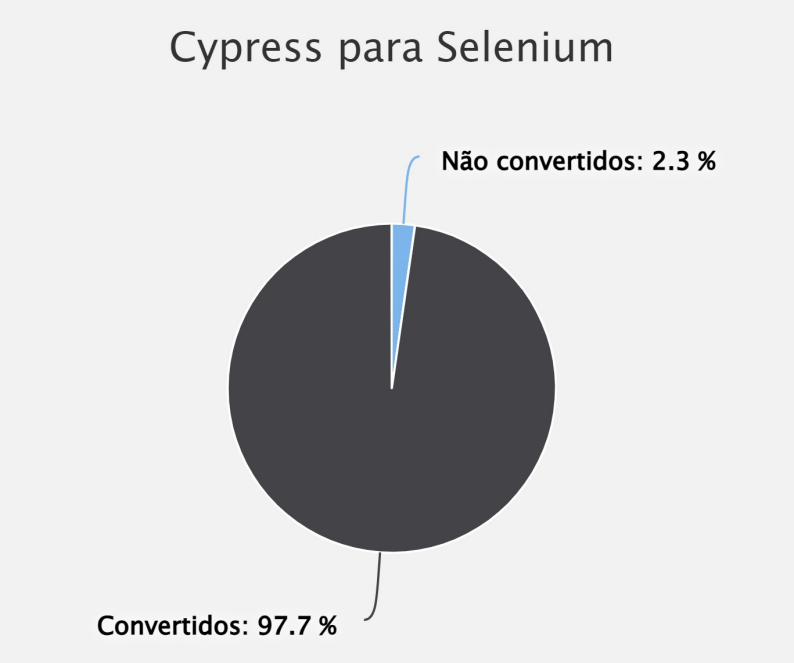

Figura 10. Dados da conversão de Cypress para Selenium

32, 20 foram executados com sucesso, isto é, executaram sem erros e produziram os mesmo resultados do *script* original, sendo equivalente a 62.5% dos testes. As outras 12 conversões falharam durante a execução, em outras palavras, nenhum teste obteve erro de compilação ou sintaxe, porém 12 *scripts* escritos em Cypress, correspondendo a 37.5%, tiveram algum resultado diferente do original em Selenium, por exemplo, um teste que passa no código inicial, obteve falha no convertido. Após analisar os 12 que falharam, constatou-se que 6 foram devido ao uso do recurso de *offset*, recurso que o Cypress não tem suporte, consequentemente, os testes que utilizaram esse recurso, após convertidos,

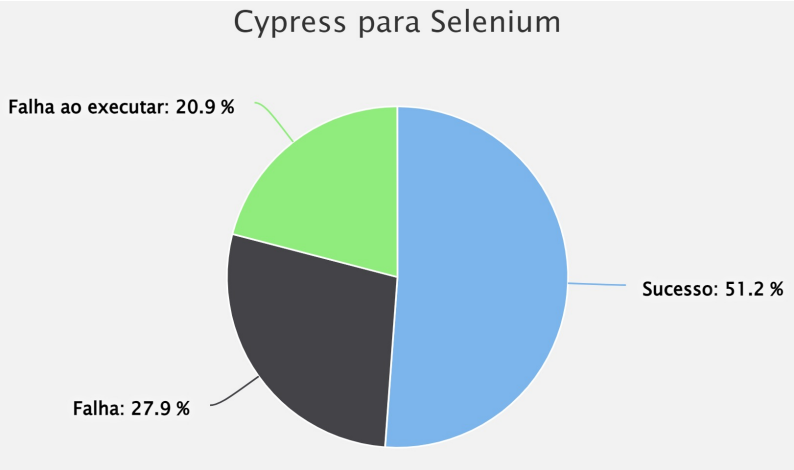

Figura 11. Dados da execução de Cypress para Selenium

levaram para um ponto diferente na tela ou davam erro. Os outros 5 *scripts* gerados falharam em razão de problemas nativos do Cypress. Por exemplo, testes que utilizam comandos que simulam ações do teclado, como a inserção de letra maiúscula com o atalho Shift + letra. No Cypress, existe um problema conhecido em que a letra inserida continua minúscula, apesar de ter sido supostamente clicada com o Shift. Mais detalhes podem ser vistos em um *issue* aberto no GitHub[2]. Nas outras falhas de execução, a substituição de algum comando simples fazia com que o resultado esperado fosse alcançado. Por exemplo, ao substituir o 'have.text' por 'include.text' no código migrado para o Cypress fez o *script* obtido funcionar da mesma forma que o *script* original em Selenium. Apenas 1 dos 12 *scripts* que falharam precisou de uma análise mais aprofundada do motivo para a falha.

Já a conversão do Cypress para Selenium, dos 44 testes em Cypress, 1 não foi convertido, ou seja, 2.27%. Isso aconteceu pois o teste original usa comandos de manipulação de *requests* e, segundo a resposta dada pelo próprio ChatGPT, necessitaria de ferramentas adicionais para funcionar no Selenium. Portanto, 43 testes foram convertidos, equivalentes a 97.73%. Desses convertidos, as taxas de sucesso e falha foram similares com a de tradução do Selenium para Cypress. 22 dos 43 testes obtidos foram executados com sucesso, indicando uma taxa de 51.16% no sucesso da execução. Dos 21 restantes, 12, portanto 27.91%, executaram e tiveram um resultado diferente do original. Para esses 12, o resultado da execução foi diferente do *script* original por problema no comando gerado para Selenium. Enquanto no código original o teste é bem-sucedido, no convertido ocorre falha. A maioria dos problemas encontrados para a não ocorrência da execução é relacionado ao caminho do elemento envolvido no teste que não foi identificado, porém, outros erros de difícil compreensão foram relatados. A maior diferença na tradução entre os dois *frameworks* se apresenta na taxa de falhas para executar. Na tradução para Cypress, todos executaram, enquanto na tradução para Selenium, 9 *scripts* não executaram, o que representa 20.93% que não executaram. Isso pode ser explicado por alguns motivos. Os principais deles é que os *scripts* da base de testes do Cypress são mais extensos (possuem

[2]`https://github.com/cypress-io/cypress/issues/2386`

mais linhas de código). Como a abordagem utilizada para a conversão foi o *single shot*, o ChatGPT pode ter tido mais dificuldade para converter *scripts* maiores. Outro fator que desfavorece a tradução para Selenium pode ser é a complexidade maior de Selenium em comparação com o Cypress.

5.2. Métricas de código entre o *script* original e o convertido

Nesta seção, métricas dos códigos de origem e de destino são observadas. Para isso, foi utilizado o SonarQube em todos os testes traduzidos, já que com essa ferramenta é possível obter resultados relacionados ao número de linhas, quantidade de *code smells*, débitos técnicos, complexidade. Essas métricas foram utilizadas na comparação entre o *script* original e o migrado. As Figuras 12 e 13 mostram esses resultados, primeiramente comparando o código Cypress obtido a partir de Selenium, em seguida, comparando o código Selenium obtido a partir de Cypress.

É relevante ressaltar que nesse caso, quando o código original era em Selenium com Python e o convertido foi para Cypress, houve uma diminuição de quase todas as métricas, apenas a complexidade ciclomática que aumentou. Esse aumento pode ser devido à conversão de *scripts* que originalmente usam *offset* para conduzir o ponteiro a um canto específico na tela, Cypress não possui eventos para lidar diretamente com ponteiros como Selenium possui, desse modo, o ChatGPT propôs algumas alternativas para simular esse comportamento que podem ter feito a complexidade ciclomática aumentar. Em contrapartida, as outras métricas diminuíram, visto que Cypress é um *framework* mais simples que Selenium, ajudando, também, na melhora da qualidade do código.

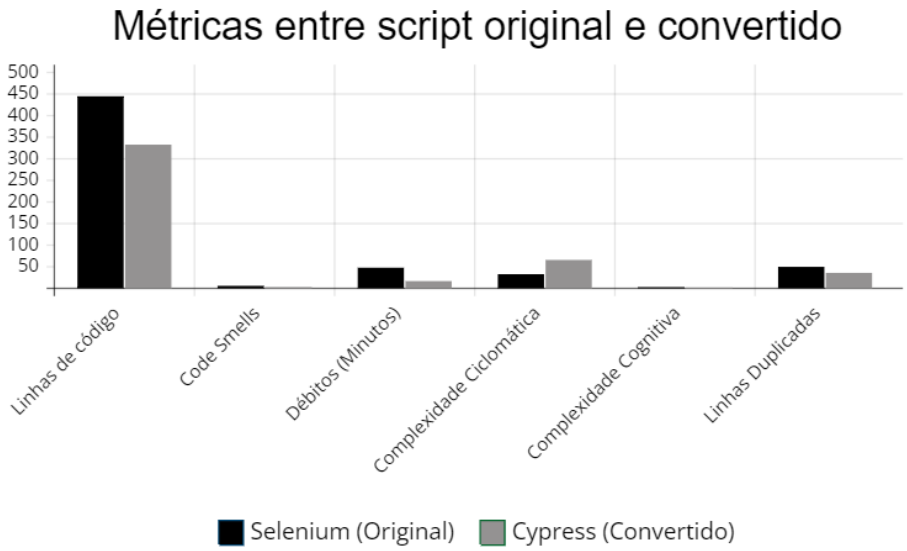

Figura 12. Comparação entre algumas métricas de *software* obtidas com o SonarQube em que o *script* original é Selenium e o convertido Cypress

Por outro lado, no caso em que o *script* original é em Cypress e o portado em Selenium, houve casos de testes que não foram convertidos, explicando a diminuição pequena no número de linhas, consequentemente, resultando também na melhora da complexidade

ciclomática. Contudo, pelas métricas do SonarQube, na verdade, houve uma piora significativa na qualidade do código, visto que aumentaram significantemente as quantidades de *code smells*, débitos, complexidade cognitiva e linhas duplicadas. Este resultado era esperado, já que o Selenium, no geral, necessita de mais comandos que o Cypress para executar as mesmas ações.

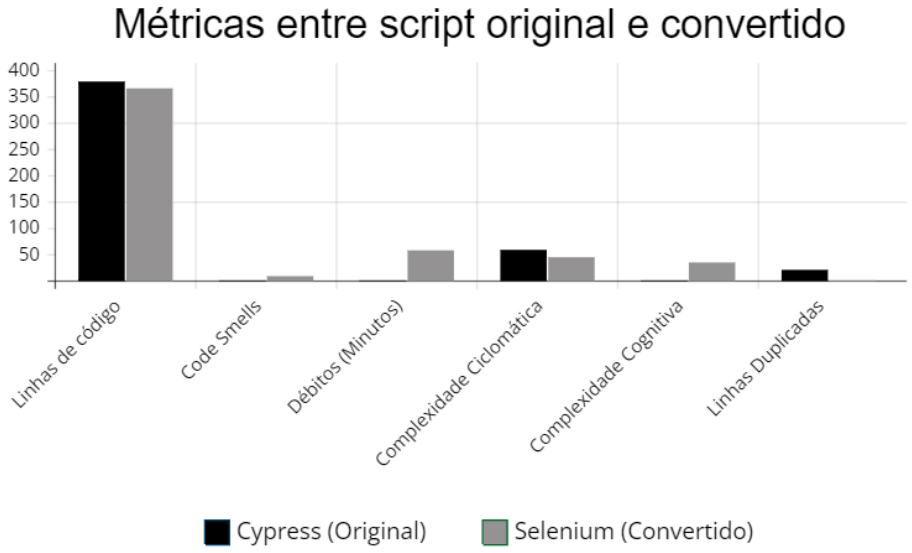

Figura 13. Comparação entre algumas métricas de *software* obtidas com o SonarQube em que o *script* original é Cypress e o convertido Selenium

5.3. Estudo comparativo

Em tempos de produção deste trabalho, não foram identificados trabalhos anteriores relacionados à conversão automática de *scripts* de teste entre diferentes *frameworks*. Todavia, alguns autores já exploraram a eficácia da tradução de código de aplicação entre diferentes plataformas. Como exemplo, há um artigo [Mahmoud et al. 2023] que propôs uma ferramenta própria de conversão, obtendo uma acurácia de 89.3% na tradução de código de aplicação IOS para Android, adicionalmente, esses mesmos autores testaram a migração utilizando o ChatGPT, resultando em acurácia de 100%. Em outro artigo [Yu et al. 2023] relacionado não relata as taxas de sucesso na conversão, mas discute os problemas encontrados durante a conversão assim como a necessidade da análise humana e correções manuais. Figura 14 introduz um gráfico comparativo ente o resultado deste trabalho com os resultados de [Mahmoud et al. 2023] é exibido abaixo.

5.4. Limitações

Como limitações deste estudo, podemos ressaltar as duas principais. A primeira é a subjetividade da IA generativa. Como ela gera respostas a partir de um *prompt* e é alimentada sempre que é usada, pode ser que ao tentar replicar o experimento seja obtido resultados diferentes (uma vez que o modelo é probabilístico). Seria necessário reproduzir a conversão dos *scripts* diversas vezes e avaliar a média das métricas obtidas para uma análise estatística mais robusta. A segunda é o *prompt* propriamente dito. A elaboração do *prompt*

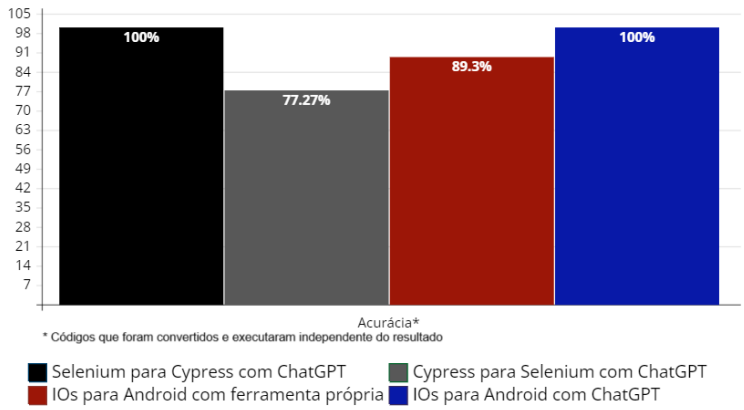

Figura 14. Dados comparativos com trabalhos anteriores

é uma etapa primordial para a obtenção de resultados de qualidade, isto dito, *prompts* diferentes do que foi usado neste trabalho pode resultar em respostas piores ou melhores. O presente trabalho não explorou variações no prompt que podem influenciar o resultado da conversão.

6. Conclusão e trabalhos futuros

O estudo da conversão de *scripts* de um *framework* para outro se mostrou útil, pois demonstrou a eficiência de usar a IA generativa para otimizar o tempo e aumentar a produtividade, demandando menos esforço para o profissional de qualidade. Essa conversão foi avaliada utilizando os dois mais populares *frameworks* de automação de testes para sistemas web da atualidade, Selenium e Cypress. Além disso, um *prompt* foi proposto para a realização desta atividade, como também a forma de utilizá-lo, sendo esta em *single shot*.

Os objetivos propostos foram alcançados e as perguntas se é possível usar uma IA generativa com o intuito de transformar *scripts* escritos em uma linguagem para outra? e qual a taxa de sucesso nessa portabilidade? foram respondidas. Os resultados demonstraram que é possível utilizar o ChatGPT para ajudar na migração, deixando claro que a análise e correções humanas precisam ser feitas, mas existe um ganho real na produtividade na realização de uma atividade de portabilidade de códigos. Quanto a taxa de sucesso, essa pergunta foi respondida de várias perspectivas. Se considerarmos sucesso como a quantidade de *scripts* convertidos, no caso de Selenium para Cypress, obtivermos um resultado de 100%, enquanto de Cypress para Selenium, 97.73%. No entanto, a interpretação mais condizente seria considerar sucesso como sendo *scripts* convertidos e na execução obter o mesmo resultado do original. Neste caso, de Selenium para Cypress foi 62.5% e Cypress para Selenium, 51.16%.Essas taxas obtidas podem não parecer satisfatórias, porém outro resultado obtido foi que em casos de falhas, algumas delas eram por motivos que foram de fácil resolução, ademais, o profissional que precisa fazer uma conversão e tem a chance de ter um código já nos moldes necessários, tem o seu trabalho facilitado.

Como trabalhos futuros, um *prompt* com mais detalhes do processo que a IA generativa deve fazer pode ser estudado. Este estudo teria como proposito aumentar a taxa

de sucesso com resultados congruentes aos códigos originais. Além disso, outra forma de alimentar a LLM com informações também pode ser investigada neste processo, descobrindo quais seriam as consequências nos dados obtidos após a conversão. Outra possibilidade é analisar a conversão com o uso de outras IA generativas, como, por exemplo, uma versão mais avançada do ChatGPT, ou até o uso do Gemini, IA do Google, ou do Copilot, IA da Microsoft, fazendo um comparativo obtido nestas outras ferramentas. Ainda como trabalho futuros, planejamos realizar um experimento mais abrangente e controlado, o que permitirá a avaliação estatística dos resultados. Por último, a forma que o *prompts* é enviado para o ChatGPT. A forma adotada aqui foi a *single shot*, se adotar outros meios, como, por exemplo, ir alimentando a LLM com informações ao pouco até resultar no pedido de conversão, pode ocasionar resultados bastante diferentes do obtido via este experimento.

Referências

[Ahmed and Sharo 2023] Ahmed, Y. A. and Sharo, A. (2023). On the education effect of chatgpt: Is ai chatgpt to dominate education career profession? In *2023 International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS)*, pages 79–84.

[Altamimi 2023] Altamimi, A. B. (2023). Effectiveness of chatgpt in essay autograding. In *2023 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pages 102–106.

[Axelsson and Skoglund 2016] Axelsson, J. and Skoglund, M. (2016). Quality assurance in software ecosystems: A systematic literature mapping and research agenda. *The Journal of systems and software*, 114:69–81.

[Caramancion 2023] Caramancion, K. M. (2023). Harnessing the power of chatgpt to decimate mis/disinformation: Using chatgpt for fake news detection. In *2023 IEEE World AI IoT Congress (AIIoT)*, pages 0042–0046.

[Cypress.io 2024] Cypress.io (2024). About us. `https://www.cypress.io/about-us`. Acesso em: 28 de jul. de 2024.

[El-Kassas et al. 2016] El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., and Wahba, A. M. (2016). Enhanced code conversion approach for the integrated cross-platform mobile development (icpmd). *IEEE Transactions on Software Engineering*, 42(11):1036–1053.

[Foster 2022] Foster, D. (2022). *Generative deep learning*. "O'Reilly Media, Inc.".

[G1 2023] G1 (2023). Disputa voto a voto, uso de ia, relação com o brasil: como milei e massa chegam ao 2º turno das eleições argentinas. `https://g1.globo.com/mundo/noticia/2023/11/19/disputa-voto-a-voto-uso-de-ia-relacao-com-o-brasil-como-milei-e-massa.ghtml`. Acesso em: 25 de jul. de 2024.

[Garousi et al. 2020] Garousi, V., Rainer, A., Lauvås Jr, P., and Arcuri, A. (2020). Software-testing education: A systematic literature mapping. *Journal of Systems and Software*, 165:110570.

[Garousi et al. 2018] Garousi, V., Özkan, R., and Betin-Can, A. (2018). Multi-objective regression test selection in practice: An empirical study in the defense software industry. *Information and Software Technology*, 103:40–54.

[GH and Badkar 2023] GH and Badkar, A. (2023). Best test automation frameworks (updated 2023). `https://www.browserstack.com/guide/best-test-automation-frameworks`. Acesso em: 26 de jul. de 2024.

[GitHub 2024a] GitHub (2024a). `https://github.com/SeleniumHQ/selenium/network/dependents`. Acesso em: 26 de jul. de 2024.

[GitHub 2024b] GitHub (2024b). `https://github.com/cypress-io/cypress/network/dependents`. Acesso em: 26 de jul. de 2024.

[GitHub 2024c] GitHub (2024c). `https://github.com/webdriverio/webdriverio/network/dependents`. Acesso em: 26 de jul. de 2024.

[GitHub 2024d] GitHub (2024d). `https://github.com/DevExpress/testcafe/network/dependents`. Acesso em: 26 de jul. de 2024.

[GitHub 2024e] GitHub (2024e). `https://github.com/nightwatchjs/nightwatch/network/dependents`. Acesso em: 26 de jul. de 2024.

[GitHub 2024f] GitHub (2024f). `https://github.com/appium/appium/network/dependents`. Acesso em: 26 de jul. de 2024.

[GitHub 2024g] GitHub (2024g). `https://github.com/cucumber/cucumber-jvm/network/dependents`. Acesso em: 26 de jul. de 2024.

[Harrell et al. 2018] Harrell, S. L., Kitson, J., Bird, R., Pennycook, S. J., Sewall, J., Jacobsen, D., Asanza, D. N., Hsu, A., Carrillo, H. C., Kim, H., and Robey, R. (2018). Effective performance portability. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 24–36.

[Huizinga and Kolawa 2007] Huizinga, D. and Kolawa, A. (2007). *Automated Defect Prevention: Best Practices in Software Management*. Wiley-Interscience.

[Ige et al. 2024] Ige, T., Kiekintveld, C., Piplai, A., Wagler, A., Kolade, O., and Hafiz Matti, B. (2024). An in-depth investigation into the performance of state-of-the-art zero-shot, single-shot, and few-shot learning approaches on an out-of-distribution zero-day malware attack detection. In *2024 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6.

[Jabrayilzade et al. 2021] Jabrayilzade, E., Gürkan, O., and Tüzün, E. (2021). Towards a taxonomy of inline code comment smells. In *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 131–135.

[Jones and Bonsignour 2011] Jones, C. and Bonsignour, O. (2011). *The economics of software quality*. Addison-Wesley Professional.

[Kabir et al. 2023] Kabir, S., Udo-Imeh, D. N., Kou, B., and Zhang, T. (2023). Who answers it better? an in-depth analysis of chatgpt and stack overflow answers to software engineering questions.

[Kinsbruner and Bahmutov 2022] Kinsbruner, E. and Bahmutov, G. (2022). *A Frontend Web Developer's Guide to Testing: Explore leading web test automation frameworks and their future driven by low-code and AI*. Packt Publishing.

[Lagstedt et al. 2021] Lagstedt, A., Dirin, A., and Williams, P. (2021). Quality practitioners' differing perspectives on future software quality assurance practices. *International journal of interactive mobile technologies*, 15(24):134–154.

[Leotta et al. 2023] Leotta, M., Ricca, F., Marchetto, A., and Olianas, D. (2023). An empirical study to compare three web test automation approaches: Nlp-based, programmable, and capture&replay. *Journal of Software: Evolution and Process*, page e2606.

[Mahmoud et al. 2023] Mahmoud, A. T., Muhammad, A. A., Yousef, A. H., Medhat, W., Zayed, H. H., and Selim, S. (2023). Compiler-based web services code conversion model for different languages of mobile application. In *2023 Intelligent Methods, Systems, and Applications (IMSA)*, pages 464–469.

[Oh and Shon 2023] Oh, S. and Shon, T. (2023). Cybersecurity issues in generative ai. In *2023 International Conference on Platform Technology and Service (PlatCon)*, pages 97–100.

[O'Reilly Media 2023] O'Reilly Media (2023). O'reilly releases 2023 generative ai in the enterprise report revealing unprecedented 67 `https://www.oreilly.com/pub/pr/3435`. Acesso em: 26 de jul. de 2024.

[Pelivani et al. 2022] Pelivani, E., Besimi, A., and Cico, B. (2022). A comparative study of ui testing framework. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–5.

[Polo et al. 2013] Polo, M., Reales, P., Piattini, M., and Ebert, C. (2013). Test automation. *IEEE Software*, 30(1):84–89.

[Ramya et al. 2017] Ramya, P., Sindhura, V., and Sagar, P. V. (2017). Testing using selenium web driver. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–7.

[Rich et al. 2009] Rich, E., Knight, K., and Nair, S. B. (2009). *Artificial Intelligence*. Tata Mcgraw Hill Education Private Limited, 3^a edition.

[Russell and Norvig 2010] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

[Siva et al. 2023] Siva, R., S, K., Hariharan, B., and Premkumar, N. (2023). Automatic software bug prediction using adaptive golden eagle optimizer with deep learning. *Multimedia tools and applications*.

[Software Freedom Conservancy 2024] Software Freedom Conservancy (2024). About selenium. `https://www.selenium.dev/about/`. Acesso em: 28 de jul. de 2024.

[Sogeti 2023] Sogeti (2023). World quality report 2023-24. `https://www.sogeti.com/explore/reports/world-quality-report-2023-24/`. Acesso em: 26 de jul. de 2024.

[SonarSource 2024] SonarSource (2024). https://docs.sonarsource.com/sonarqube/latest/. Acesso em: 21 de set. de 2024.

[Tahvili et al. 2018] Tahvili, S., Afzal, W., Saadatmand, M., Bohlin, M., and Ameerjan, S. H. (2018). Espret: A tool for execution time estimation of manual test cases. *Journal of Systems and Software*, 146:26–41.

[Tiobe 2025] Tiobe (2025). https://www.tiobe.com/tiobe-index/. Acesso em: 08 de abr. de 2025.

[Wang 2018] Wang, Y. (2018). Test automation maturity assessment. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 424–425.

[Wiklund et al. 2017] Wiklund, K., Eldh, S., Sundmark, D., and Lundqvist, K. (2017). Impediments for software test automation: A systematic literature review. *Software Testing, Verification and Reliability*, 27(8):e1639. e1639 stvr.1639.

[Yu et al. 2023] Yu, S., Fang, C., Ling, Y., Wu, C., and Chen, Z. (2023). Llm for test script generation and migration: Challenges, capabilities, and opportunities. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, pages 206–217.