2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW) | 979-8-3503-3032-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASEW60602.2023.00028

GUI-Based Software Testing: An Automated Approach Using GPT-4 and Selenium WebDriver

Daniel Zimmermann
Software Engineering (SE)
FZI Research Center for Information Technology
Karlsruhe, Germany
daniel.zimmermann@fzi.de

Anne Koziolek
KASTEL - Institute of Information Security and Dependability
Karlsruhe Institute of Technology
Karlsruhe, Germany
anne.koziolek@kit.edu

***Abstract*—This paper presents a novel method for GUI testing in web applications that largely automates the process by integrating the advanced language model GPT-4 with Selenium, a popular web application testing framework. Unlike traditional deep learning approaches, which require extensive training data, GPT-4 is pre-trained on a large corpus, giving it significant generalisation and inference capabilities. These capabilities allow testing without the need for recorded data from human testers, significantly reducing the time and effort required for the testing process. We also compare the efficiency of our integrated GPT-4 approach with monkey testing, a widely used technique for automated GUI testing where user input is randomly generated. To evaluate our approach, we implemented a web calculator with an integrated code coverage system. The results show that our integrated GPT-4 approach provides significantly better branch coverage compared to monkey testing. These results highlight the significant potential of integrating specific AI models such as GPT-4 and automated testing tools to improve the accuracy and efficiency of GUI testing in web applications.**

***Index Terms*—UI Testing, Test Automation, GPT-4, Language Models**

I. Introduction

The rapid advance of digitalization underscores the importance of efficient testing for GUI-based systems. The inherent complexity and dynamic quality of GUIs challenge the implementation of thorough testing, especially when compared to unit testing [1], [2]. These challenges open opportunities for advanced AI models to improve the effectiveness of these tests [3]–[5]. To address these challenges, we propose an innovative approach that leverages the advanced language model GPT-4 to significantly improve software testing coverage.

The field of AI has recently witnessed transformative developments, resulting in techniques that require less domain-specific training data [6]–[9]. GPT-4 stands out in this field as a powerful and versatile model that excels at various tasks without much guidance. Such capabilities open up new horizons for automation in end-to-end GUI testing in software testing [10], [11]. By integrating the Selenium WebDriver - a predominant tool for automating browser actions - with the exceptional reasoning capabilities of GPT-4, our method eliminates the reliance on pre-recorded human interaction data for the application under test.

Our approach exploits the inherent structure of web applications. Their HTML content is translated directly into textual representations that can be processed by language models such as GPT-4. Combined with automated testing tools like Selenium, the process is notably efficient. Due to their complexity and diverse user interaction possibilities, web applications are ideal for evaluating AI models in automated testing scenarios. Our test results[1], scripts[2], and sample web applications[3] are online, which ensures reproducibility of our findings and promotes transparency.

As a running example, we applied our methodology to a web-based calculator application developed in HTML. In addition to basic arithmetic functions, this calculator application allows advanced unary and binary functions to be performed using only left mouse clicks. Avoiding exhaustive testing, we integrated a code coverage system into the calculator application. We then compared our method with monkey testing, a widely used automated GUI testing technique [12], focusing on branch coverage as the primary evaluation metric. Our preliminary results suggest that our approach outperforms monkey testing in terms of branch coverage, thus obviating the need for training data from human testers.

Compared to our previous study with GPT-3 [11], this paper highlights the advances achieved through GPT-4 integration. The main distinctions of this research are:

1) **Independence from human interaction data**: Using GPT-4's enhanced reasoning, we test without relying on data recorded by human testers.
2) **Web Application Testing**: We perform our tests on a web application using the Selenium WebDriver.
3) **Testing coverage**: Our focus shifted to maximising branch coverage beyond the execution of rudimentary human-created test cases.

In essence, our study aims to explore the synergy between advanced AI language models such as GPT-4 and automated testing tools, in the hope of achieving greater accuracy and efficiency in GUI testing.

[1]https://doi.org/10.5281/zenodo.8167446
[2]https://github.com/SoftwareTestingLLMs/WebtestingWithLLMs
[3]https://github.com/SoftwareTestingLLMs/SampleWebsitesForTesting

2151-0849/23/$31.00 ©2023 IEEE
DOI 10.1109/ASEW60602.2023.00028

II. Related Work

Past research in automated testing of GUI-based systems has largely concentrated on developing tools like Selenium and Appium[4]. These tools provide support for interacting with GUI elements. While tools such as Selenium WebDriver require test cases to be written in specific programming languages, there are other solutions like Selenium IDE that offer record-and-play features, eliminating the need for programming knowledge[5]. Nonetheless, the automated tests, especially those created with more technical tools, can be brittle and hard to maintain due to changes in the application's GUI elements.

Deep learning techniques have recently been incorporated in the field of automated GUI testing. Daragh and Malek present a tool, Deep GUI, that uses deep learning to generate GUI inputs [12]. Their black-box approach does not require knowledge of the implementation details of the app under test. They build a model of valid GUI interactions based on app screenshots, which drives the generation of intelligent inputs. By integrating their tool into Google Monkey, they developed Monkey++, which demonstrated improved performance in applications requiring complex GUI interactions. Although they noted limitations in their approach, such as the time-consuming data collection process and the lack of consideration of previous interaction history with the application, Deep GUI establishes an important step towards a fully black-box and cross-platform test input generation tool.

Another contribution is the DRIFT framework proposed by Harries et al. [13]. DRIFT uses Q-learning and Graph Neural Networks to operate on a symbolic representation of a user interface for efficient GUI testing. Unlike other automated GUI testing tools, which require lengthy testing execution times, DRIFT employs deep reinforcement learning to significantly streamline software testing processes.

In addition, Khaliq et al. proposed a deep learning-based framework for automating functional UI testing [14], [15]. They used object and text detection algorithms to automatically identify elements from a UI, and text-generation transformers to convert the UI element descriptions into test designer-written test cases.

Despite these advancements, many automated GUI testing methods remain heavily dependent on manual input, are complex for non-technical users to operate, and are brittle due to changes in the applications' GUI elements. Notably, our approach does not require intensive training: we neither need exhaustively created training data nor time-consuming training processes often associated with reinforcement learning techniques.

III. Proposed Approach

Our approach to automating GUI testing takes a unique route, leveraging the extensive pre-training and impressive generation capabilities of GPT-4. This route bypasses the hassle of fine-tuning models with pre-recorded user interaction data from human testers, obviating the necessity of this resource-intensive step.

[4]https://appium.io/

[5]https://www.selenium.dev/selenium-ide/

TABLE I
Context sizes of different large language models

Model	Max Context Size (tokens)
Fine-tuned GPT-3	2048
GPT-3	4096
ChatGPT	4096
GPT-4 (8k)	8192
GPT-4 (32k)	32768

The integration of GPT-4 with Selenium WebDriver underpins our approach. GPT-4 creates and interprets descriptions of the state of GUI elements of a web application under test (AUT), simulating user interactions. When an interaction with the AUT changes its state, this is reflected in the Document Object Model (DOM) maintained by WebDriver. We feed this updated state back into GPT-4, closing the feedback loop and enabling comprehensive end-to-end testing without needing human interaction.

A main limitation of GPT-4 is its capacity for remembering prior prompts and token size when testing complex web applications with numerous GUI elements. We address this by including the past interaction history and the current state in every input prompt step and recycling previous interactions based on their relevance to the updated state, as demonstrated in Figure 1.

One aspect of the approach to ensure that our tests are practical and meaningful involves considering the context sizes of large language models while generating our input prompt. As shown in Table I, different large language models have different maximum context sizes, and we designed our input prompts to ensure that they do not exceed these. The context size is particularly important in GUI testing as keeping track of past interactions is key to generating meaningful new interactions.

The increasing context sizes in the evolution of transformer architectures have largely mitigated this limitation, allowing for more extensive interaction history [6], [7]. Nevertheless, we acknowledge that more complicated applications might still pose challenges, especially for cases where long-term dependencies are critical.

IV. Experimental Setup

To validate our proposed approach, we employed a custom-built web application: an HTML-based calculator as depicted in Figure 2. Although the functionality of the calculator is simple, it offers a diverse range of user interface actions. It features number buttons from 0 to 9 and basic arithmetic operations (addition, subtraction, multiplication, and division). Furthermore, it supports advanced functionalities like exponentiation and square root operations. Interaction with the calculator is processed through left mouse-clicks. Additionally, to provide a comprehensive testing environment, the calculator application integrates a built-in coverage system that tracks

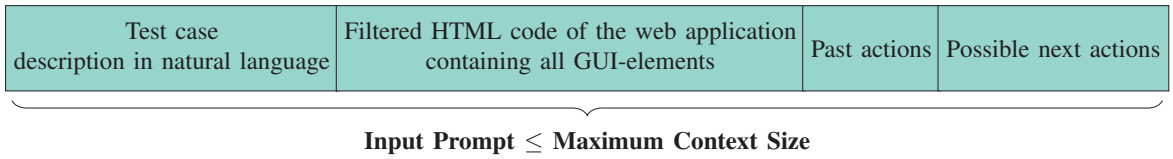

Fig. 1. Illustration of the proposed approach which leverages GPT-4's textual comprehension capabilities to simulate user interactions with the Application Under Test (AUT). The input prompt is tailored to comply with the model's maximum context size, thereby enabling GPT-4 to fully understand the entire context of the testing scenario. This approach compensates for the model's inability to reference information from earlier prompts.

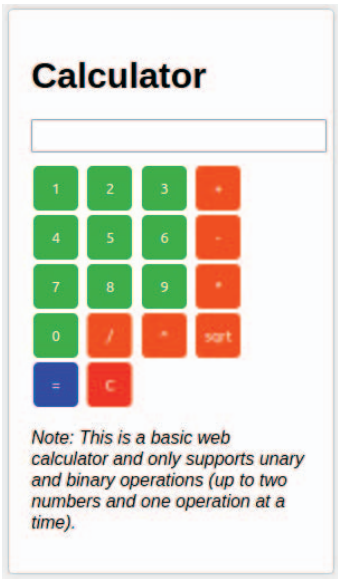

Fig. 2. Screenshot of the web-based calculator used as an application to test in this paper

each feature's usage during the testing process. It also provides an output of the branch coverage percentage, acting as the principal metric for our evaluation process.

For the operation of GUI testing, Selenium WebDriver is instrumental. The WebDriver manages the interaction with the calculator by recording HTML snapshots of its state, identifying potential interaction targets, and tracking changes in the state of the Document Object Model (DOM) after each interaction. By feeding this updated state back into the GPT-4 model, a closed-loop feedback system is established.

Our testing environment makes use of the function-calling[6] capability made available by the OpenAI API. This capability allows the models to detect and respond with a JSON object when a function needs to be called, based on the user's input. The returned structured data can then be directly integrated into our testing script, thereby improving the reliability of interactions with the AI models.

In the testing process, the raw HTML code of the calculator web page is processed and unnecessary elements such as 'script' and 'style' code are removed before passing it to the AI model. Since the model has no access to the JavaScript source code of the application, this setup can be considered as black-box testing without knowing the internals of the Application Under Test (AUT). The cleaned HTML code guides the AI model by providing an overview of current views and interaction possibilities.

To mitigate the context size limitations of GPT-4, we designed our approach to include the past interactions of the model and the current state of the web application in every input prompt. This design ensures the AI has full context of the whole testing history for each interaction.

It's also worth noting that the same testing script is used for both the Monkey Tester and the AI model, enabling both to interact with the application using the same Selenium WebDriver session. This not only ensures fairness in comparison but also maintains an easier and interpretable experimental setup.

V. Experimental Results

Table II shows the results of the experimental comparison of the Monkey Tester, GPT-3.5-Turbo, and our GPT-4 integrated approach. All experiments were run locally on a desktop computer, and the total computation time was about 2 hours. The cost of accessing GPT-3.5-Turbo and GPT-4 via the OpenAI API for all tests, including multiple repetitions of experiments, was approximately €50.

We evaluated the models with different interaction steps per test run, specifically 10, 20, 30, 50, and 100, and ran five trials for each setup. From the initial configuration, which permitted 10 interaction steps per trial, GPT-4 consistently showed superior performance, achieving an average branch coverage of 58.82% across all trials. Conversely, the Monkey Tester and GPT-3.5-Turbo achieved maximum coverages of 47.06% and 52.94%, respectively, under the same conditions.

As we increased the number of interactions allowed per test run, the performance gap between the GPT-4 model and the other two candidates widened even further. With 20 interaction steps allowed per test run, the Monkey Tester already leveled off and registered a maximum coverage of 41.18%. GPT-3.5-Turbo performed slightly better, achieving a maximum coverage of 64.71%. In contrast, our GPT-4 model significantly outperformed the other two, registering a base coverage of 70.59%, an average coverage of 77.65%, and a maximum coverage of 82.35%.

These trends continued as we increased the number of interactions allowed in the test run to 30, 50, and finally

[6]https://platform.openai.com/docs/guides/gpt/function-calling

TABLE II
COMPARISON OF BRANCH COVERAGES AMONG MONKEY TESTER, GPT-3.5-TURBO, AND GPT-4

Number of interaction steps per test run (only left mouse clicks)	**Monkey Tester** Min Coverage	Mean Coverage	Max Coverage	**GPT-3.5-Turbo** Min Coverage	Mean Coverage	Max Coverage	**GPT-4** Min Coverage	Mean Coverage	Max Coverage
10	23.53%	**32.94%**	47.06%	23.53%	**40.0%**	52.94%	58.82%	**58.82%**	58.82%
20	23.53%	**31.76%**	41.18%	47.06%	**56.47%**	64.71%	70.59%	**77.65%**	82.35%
30	47.06%	**57.65%**	64.71%	52.94%	**58.82%**	70.59%	82.35%	**88.24%**	94.12%
50	35.29%	**56.47%**	70.59%	52.94%	**58.82%**	70.59%	82.35%	**84.7%**	94.12%
100	64.71%	**74.12%**	82.35%	70.59%	**75.29%**	82.35%	88.24%	**91.77%**	100.0%

100. In all of these tests, GPT-4 consistently outperformed both alternatives. Remarkably, GPT-4 achieved 100.0% branch coverage when given a maximum of 100 interaction steps per test run. Faced with the same challenge, the Monkey Tester only achieved a peak coverage of 82.35%. Similarly, the most advantageous performance of GPT-3.5-Turbo could only achieve an average coverage of 75.29%.

Comparative analysis of the results clearly demonstrated the superior performance of the GPT-4 model in terms of higher consistent coverage and faster coverage achievement. In contrast, the Monkey Tester and GPT-3.5-Turbo struggled to keep pace with the GPT-4 model in both maximum achievable coverage and performance consistency.

VI. CONCLUSION AND FUTURE WORK

This research has successfully demonstrated the potency of integrating artificial intelligence models, in particular GPT-4, into the process of GUI testing of web applications. Our experimental design and comparative evaluation against the monkey testing methodology revealed that the proposed approach significantly outperformed the monkey tester in terms of branch coverage.

While we have demonstrated promising results, our approach has the potential for further improvements. In future work, we plan to extend our methodology to handle more complex web applications with diverse and sophisticated GUI elements. In addition to the increased complexity of test applications, we aim to extend our approach beyond its current focus on code coverage. By developing the ability to identify defects and generate autonomous explanations for test sequences, we aim to provide developers with a deeper understanding of the testing process, provide more meaningful insights, and thus improve the overall quality of software development. We also expect to extend the applicability of our methodology to other aspects of software testing, such as usability testing. Finally, we aim to take advantage of future developments in AI models, in particular the visual model expected in GPT-4[7]. A shift to high-dimensional visual input for GUI testing could extend the applicability of our approach to mobile and desktop application testing.

Together, such enhancements could foster a new era of efficient and effective software testing, harnessing the power of AI models to reduce the time and effort associated with the testing process while increasing accuracy and coverage.

[7]https://openai.com/research/gpt-4

REFERENCES

[1] M. Leotta, B. García, F. Ricca, and J. Whitehead, "Challenges of End-to-End Testing with Selenium WebDriver and How to Face Them: A Survey," in *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. Dublin, Ireland: IEEE, Apr. 2023, pp. 339–350.

[2] N. Sunman, Y. Soydan, and H. Sözer, "Automated Web application testing driven by pre-recorded test cases," *Journal of Systems and Software*, vol. 193, p. 111441, Nov. 2022.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778.

[5] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 6517–6525.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[7] OpenAI, "GPT-4 Technical Report," Mar. 2023.

[8] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of Artificial General Intelligence: Early experiments with GPT-4," Apr. 2023.

[9] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A Comprehensive Survey on Transfer Learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021.

[10] Z. Liu, C. Chen, J. Wang, M. Chen, B. Wu, X. Che, D. Wang, and Q. Wang, "Chatting with GPT-3 for Zero-Shot Human-Like Mobile Automated GUI Testing," May 2023.

[11] D. Zimmermann and A. Koziolek, "Automating GUI-based Software Testing with GPT-3," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. Dublin, Ireland: IEEE, Apr. 2023, pp. 62–65.

[12] F. YazdaniBanafsheDaragh and S. Malek, "Deep GUI: Black-box GUI Input Generation with Deep Learning," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Melbourne, Australia: IEEE, Nov. 2021, pp. 905–916.

[13] L. Harries, R. S. Clarke, T. Chapman, S. V. P. L. N. Nallamalli, L. Ozgur, S. Jain, A. Leung, S. Lim, A. Dietrich, J. M. Hernández-Lobato, T. Ellis, C. Zhang, and K. Ciosek, "DRIFT: Deep Reinforcement Learning for Functional Software Testing," Jul. 2020.

[14] Z. Khaliq, S. U. Farooq, and D. A. Khan, "Artificial Intelligence in Software Testing : Impact, Problems, Challenges and Prospect," Jan. 2022.

[15] ——, "Transformers for GUI Testing: A Plausible Solution to Automated Test Case Generation and Flaky Tests," *Computer*, vol. 55, no. 3, pp. 64–73, Mar. 2022.