2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) | 979-8-3503-3335-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSTW58534.2023.00022

Automating GUI-based Software Testing with GPT-3

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 introduces a new method for GUI-based software testing that utilizes GPT-3, a state-of-the-art language model. The approach uses GPT-3's transformer architecture to interpret natural language test cases and programmatically navigate through the application under test. To overcome the memory limitations of the transformer architecture, we propose incorporating the current state of all GUI elements into the input prompt at each time step. Additionally, we suggest using a test automation framework to interact with the GUI elements and provide GPT-3 with information about the application's current state. To simplify the process of acquiring training data, we also present a tool for this purpose. The proposed approach has the potential to improve the efficiency of software testing by eliminating the need for manual input and allowing non-technical users to easily input test cases for both desktop and mobile applications.**

***Index Terms*—UI Testing, Test Automation, Deep Learning, Language Models**

I. Introduction

The software testing process is essential for ensuring the quality and reliability of software applications [1], [2]. However, manual testing can be time-consuming, prone to human error, and require specialized knowledge and skills to navigate through the application and execute test cases [3]. On the other hand, automated software testing can be accomplished using various methods, such as a GUI-based test automation framework [4], [5]. Nevertheless, this approach requires programming language knowledge and can be time-consuming to update after changes in the application.

To address the challenges of both manual and automated software testing, we propose utilizing GPT-3, a state-of-the-art language model [6]. GPT-3 can interpret natural language test cases, navigate through the application under test, and understand context through its transformer architecture, making it potentially suitable for GUI-based applications [7]. Additionally, GPT-3 is a general-purpose model that can learn quickly and could be adapted to multiple different software applications. By providing GPT-3 with the current state of the application under test and available actions at each time step, it can output the next action that should be taken.

We demonstrate the feasibility of our proposed method by providing a sample application and a corresponding dataset[1], along with a tool that simplifies dataset acquisition[2]. Our approach can improve the efficiency of software testing by enabling non-technical users to input test cases and eliminating the need for manual input [8]. Additionally, it is a flexible and cost-effective approach for software testing as it can be applied to a wide range of software applications. GPT-3's context-aware processing of natural language inputs and its ability to quickly learn from new tasks make it an excellent candidate for GUI-based software testing, and it can be used for black-box testing of software applications without accessing the source code of the application.

II. Related Work

Previous research in the field of automated testing of GUI-based applications has primarily focused on developing tools such as Selenium[3] and Appium[4]. These tools offer support for interacting with GUI elements, but require test cases to be written in a specific programming language and can be challenging for non-technical users to use. Additionally, the automated tests created by these tools can be fragile and difficult to maintain due to changes in the GUI elements of the application.

Recently, deep learning techniques have been applied to automated testing, such as the reinforcement learning framework DRIFT proposed by Harries et al. [9]. This framework operates on the symbolic representation of the user interface and uses Q-learning through Batch-RL to model the state-action value function with a Graph Neural Network. The authors applied DRIFT to testing the Windows 10 operating system and demonstrated that it can robustly trigger the desired software functionality in a fully automated manner.

Khaliq et al. also proposed a deep learning-based framework for automating the process of functional UI testing [10], [11]. They employed object and text detection algorithms to obtain elements from an application UI automatically and

[1]https://doi.org/10.5281/zenodo.7688700
[2]https://github.com/neuroevolution-ai/SoftwareTestingLanguageModels
[3]https://www.selenium.dev/
[4]https://appium.io/

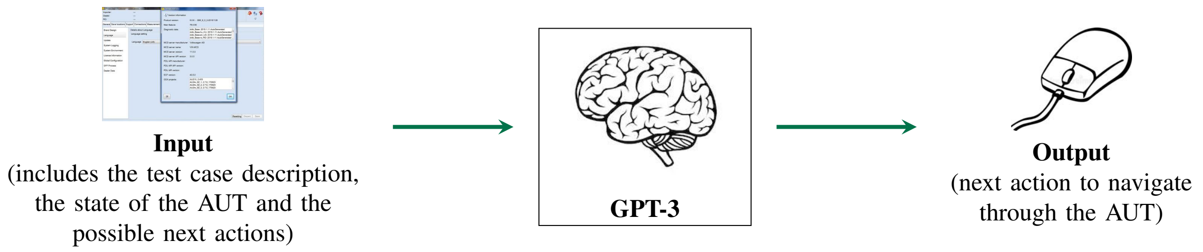

Input
(includes the test case description, the state of the AUT and the possible next actions)
GPT-3
Output
(next action to navigate through the AUT)

Fig. 1: The proposed approach utilizes deep neural network-based agents, specifically GPT-3, to simulate user interactions with the application under test (AUT) through test case descriptions in natural language.

used text-generation transformers to transform the UI element description into actual test designer-written test cases. The generated test cases were then translated into executable test scripts using a simple parser.

In summary, while previous approaches have been presented for GUI-based automated testing, they have limitations such as the need for manual input and difficulty for non-technical users to use. Our approach of utilizing GPT-3 aims to improve the efficiency of software testing by reducing the effort for manual input. Additionally, the ability to train a single model that can be applied to multiple software applications can significantly decrease the time and effort needed for retraining.

III. Proposed Method

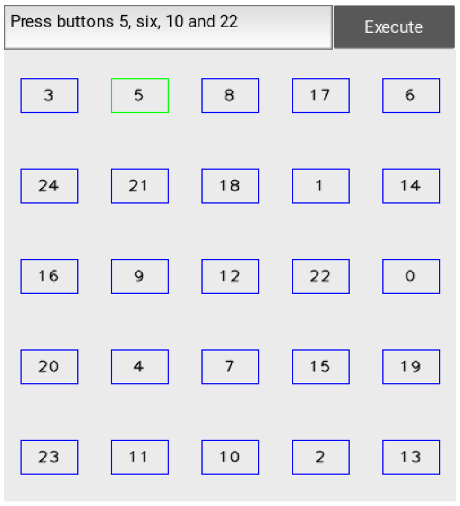
We propose incorporating GPT-3 into the process of GUI-based software testing to improve efficiency. Our approach involves providing GPT-3 with natural language test cases and utilizing its capabilities to navigate through the software and execute the test case. GPT-3 is provided with the test case description in natural language, the current state of the Application Under Test (AUT), as well as the available actions at each time step in its input prompt. It then outputs the next action that should be taken, allowing GPT-3 to programmatically traverse the AUT and execute test cases.

One of the key advantages of GPT-3 for this approach is its transformer architecture, which employs a self-attention mechanism to process input text. This allows the model to better understand natural language descriptions of test cases, helping it identify the desired actions to take in the AUT. Additionally, GPT-3 can be fine-tuned to recognize the domain-specific language, allowing it to better understand the context of the test case and the AUT. However, GPT-3's transformer architecture does not retain previous input prompts during interactions with the software. Unlike traditional recurrent neural networks (RNNs) that maintain an explicit memory or state of past inputs, the transformer architecture processes the input prompt as a single sequence. This can result in the model losing track of the already-executed parts of the test case and repeating steps. To address this limitation, we propose incorporating the current state of all GUI elements in the AUT that have already been visited into the input prompt at every step. However, due to GPT-3's limited context size, it may not be suitable for complex applications with many GUI elements. Nevertheless, advancements in transformer architectures may lead to larger context sizes, making GPT-3 and other advanced language generation models more efficient for software testing in the future.

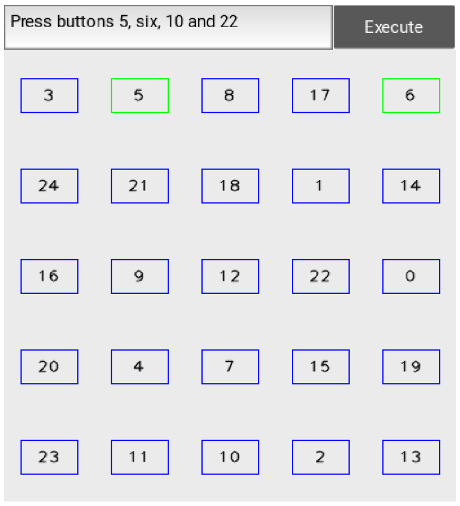
In the sample application discussed in this paper, we can directly access the state and provide GPT-3 with the necessary information about the current state of the application, as well as feed the application with actions. However, to fully utilize GPT-3 for a wide range of different software applications, we suggest using a test automation framework, like Selenium. This framework can interact with the graphical user interface (GUI) elements of the AUT and provide GPT-3 with the required information. Additionally, the framework can execute actions specified by GPT-3, allowing GPT-3 to interact with the AUT without any changes to the application. If a test automation framework is not available, we recommend using image processing tools such as OpenCV[5] or convolutional deep neural networks to interact with the GUI elements of the AUT and identify the current state of the application.

To effectively fine-tune the model, we suggest using a dataset composed of natural language test cases that detail the steps for completing a specific task. The dataset should be large enough to ensure the model's ability to generalize to new tasks and should also include a diverse range of test cases that cover various scenarios and use cases. Additionally, the dataset should contain relevant information about the AUT's state at each step in the test case. This information can aid the model in understanding the current state of the AUT and the actions that can be taken at each step. To simplify the process of acquiring training data, we have developed a tool specifically for this purpose. The tool enables users to input test case descriptions in natural language and records the data as the user executes the specific test case. It also captures information about the current state of the application and the available actions. This data can then be used to fine-tune GPT-3 to navigate the AUT and execute test cases. Once the model is fine-tuned, it can be applied to the AUT to programmatically navigate the application and execute test cases. GPT-3's ability to process natural language enables non-technical users to input test cases

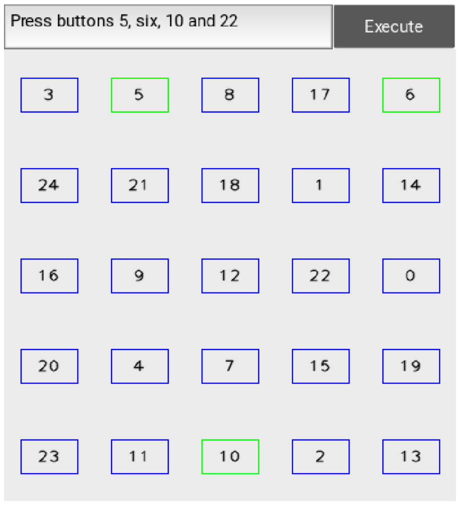
[5]https://opencv.org//


Press buttons 5, six, 10 and 22

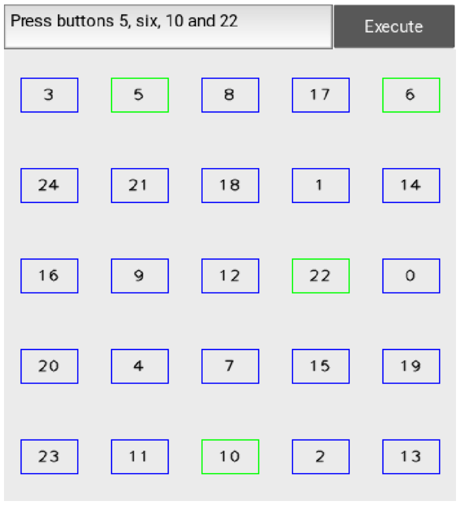
(a) Step 1


Press buttons 5, six, 10 and 22

(b) Step 2


Press buttons 5, six, 10 and 22

(c) Step 3


Press buttons 5, six, 10 and 22

(d) Step 4

Fig. 2: Screenshots of the sample GUI application programmatically executing a test case

and execute them without any prior knowledge of the syntax or commands used to interact with the software and without to need to learn the technical details of the application.

The utilization of fine-tuning a single model for multiple software applications can significantly reduce the time and resources required for retraining, as well as lower the cost of maintaining multiple models, making it more cost-effective for organizations. Furthermore, by using a single model for multiple applications, the training efficiency is improved by leveraging a larger and more diverse dataset. This approach can also be utilized to evaluate new versions of an application without the need for retraining, as GPT-3 can adapt to changes.

IV. Experimental Results

To demonstrate the feasibility of our proposed approach, we developed a sample application that utilized a grid of 25 buttons for testing with GPT-3. The application required the GPT-3 agent to click the buttons in a specific order based on the corresponding test case description. We compiled a dataset of 144 samples to train the model, which included descriptions of various test cases such as pressing a couple of buttons in a specific order or systematically pressing multiple buttons. The dataset also included the current state of the application, allowing the GPT-3 agent to consider past actions. We split the dataset into a training set (115 samples) and a validation set (29 samples) and fine-tuned the GPT-3 model with the training set using the largest DaVinci model with 175 billion parameters.

TABLE I: Wrong predictions on the validation set

Input Prompt	GPT-3 Prediction	Correct Prediction
Press all buttons in ascending order but skip all prime numbers State: {'pressed buttons': [0, 1]}	2	4
Press all buttons in descending order State: {'pressed buttons': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]}	1	0
Press the buttons in ascending order beginning with 1 skipping the buttons, 4, 5, 8 and 21 State: {'pressed buttons': [1, 2, 3, 6, 7]}	8	9

After fine-tuning, we evaluated the trained model on the validation set and found that it correctly predicted 26 out of 29 samples, with 3 incorrect predictions. The details of the incorrect predictions are shown in Table I, which provides a comprehensive view of the test cases in the validation set that the model predicted incorrectly, along with the expected and predicted outputs. Each row in the table corresponds to one wrongly predicted test case. The first column represents the input prompt, which includes the natural language test case description and the current state of the application. The second column shows the model's prediction, while the third column displays the correct prediction. For instance, in the first row of the table, the GPT-3 model predicted 2 as the next action, while the correct prediction was 4 since the test case required pressing buttons in ascending order while skipping prime numbers, and according to the state, buttons 0 and 1 were already pressed, so the next non-prime button should have been 4.

We acknowledge that using 115 samples for training is a relatively small dataset, and doubling the dataset size usually leads to a linear increase in model quality according to OpenAI's documentation[6]. Therefore, we believe that a larger dataset can further improve the agent's performance.

We further implemented the trained model in our tool and found that it was able to execute test cases with minimal error. For example, in Figure 2, the user inputs the test case in natural language in the upper text box, such as "Press buttons 5, six, 10 and 22". Then, by simply hitting the Execute button, the agent automatically executes the test case by hitting all the corresponding buttons as described in the test case. We deliberately used words instead of numerical symbols in some of the test cases' input prompts to test the model's ability to parse and understand natural language input. This allowed us to test how well the model can interpret the meaning of the input based on context and syntax rather than relying solely on the presence of numerical symbols.

V. Conclusion

In this paper, we proposed using GPT-3 for GUI-based software testing. We demonstrated that GPT-3's transformer architecture can be applied to interpret natural language test cases and navigate through the application under test. To tackle the memory limitations of the transformer architecture, we proposed incorporating the current state of all GUI elements into the input prompt at each time step. To make the process of acquiring training data simpler, we also introduced a tool. We evaluated the trained model on a sample application and the results showed that the model was able to complete test cases with minimal error, showing the feasibility of using GPT-3 for software testing.

In the future, we aim to investigate ways to further enhance the accuracy of GPT-3 for software testing. This can be achieved by increasing the amount of training data and introducing more complex test cases. In addition, we intend to create a more robust test automation framework that can interact with GUI elements and provide GPT-3 with information about the application's current state. We believe that our proposed approach has the potential to improve the efficiency of software testing by eliminating the need for manual scripting and adapting test cases. This can reduce the amount of time required to execute test cases and allow organizations to focus their resources on developing higher-quality applications.

[6]https://beta.openai.com/docs/guides/fine-tuning

References

[1] B. Beizer, *Software Testing Techniques*. Dreamtech, 2003.

[2] I. Sommerville, *Software Engineering*, 9th ed. USA: Addison-Wesley Publishing Company, 2010.

[3] C. Kaner, J. Falk, H. Q. Nguyen, and H. Q. Nguyen, *Testing Computer Software*, 2nd ed. New York Weinheim: Wiley, 1999.

[4] R. Black, *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*, 2nd ed. New York: Wiley, 2002.

[5] W. E. Lewis, *Software Testing and Continuous Quality Improvement*. Auerbach publications, 2004.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. 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. M. 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 *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[8] A. A. Sawant, P. H. Bari, and P. Chawan, "Software testing techniques and strategies," *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, no. 3, pp. 980–986, 2012.

[9] 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.

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

[11] ——, "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.