AI-Powered Software Testing: The Impact of Large Language Models on Testing Methodologies

Vahit Bayrı
Technology Architecture
KoçSistem
Istanbul, Türkiye
vahit.bayri@kocsistem.com.tr

Ece Demirel
Technology Architecture
KoçSistem
Istanbul, Türkiye
ece.demirel@kocsistem.com.tr

2023 4th International Informatics and Software Engineering Conference (IISEC) | 979-8-3503-1803-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/IISEC59749.2023.10391027

***Abstract*—Software testing is a crucial aspect of the software development lifecycle, ensuring the delivery of high-quality, reliable, and secure software systems. With the advancements in Artificial Intelligence (AI) and Natural Language Processing (NLP), Large Language Models (LLMs) have emerged as powerful tools capable of understanding and processing natural language texts easly. This article investigates the application of AI-based software testing, with a specific focus on the impact of LLMs in traditional testing methodologies. Through a comprehensive review of relevant literature and SeturDigital's 25 year testing experience, this article explores the potential benefits, challenges, and prospects of integrating LLMs into software testing.**

Keywords*—*artificial intelligence, large language model, software testing life cycle, software testing

I. Introduction

With the advancement of algorithms and methodologies, artificial intelligence (AI) has proven useful in a variety of fields. The rapid development of technology has made AI tools from theoretical possibilities into tangible realities. Artificial intelligence has several advantages in many software-related areas. Software testing is the procedure and method used to make sure that the program is error-free and to determine whether the actual results of the software match the expected outcomes by the requirements and specifications [1] .

Software testing is essential to identify defects, vulnerabilities, and inconsistencies in software systems before they are deployed to end users. Traditionally, testing methodologies have relied on manual efforts by human testers or static rule-based programs, which can be time-consuming and resource intensive. The rise of AI and LLMs has opened new possibilities for automating various aspects of the testing process, enhancing efficiency, and improving overall software quality.

One of these ground-breaking inventions is ChatGPT which was introduced by OpenAI in late 2022, a cutting-edge language model that highlights the enormous advancements made in artificial intelligence. Our goal is to make use of ChatGPT's features for software testing. We seek to improve the efficiency of our software development by including this innovative model in the testing procedure, ensuring consistent and secure performance across individual code units.

TABLE I. Performance of LLMs in Unit Testing

Dataset	Correctness	Coverage	LLM
5 Java projects from Defects4J	16.21%	5%-13% (line coverage)	Bart
10 Jave projects	40%	89% (line coverage), 90% (branch coverage)	ChatGPT
CodeSearchNet	41%	N/A	ChatGPT
HumanEval	78%	87% (line coverage), 92% (branch coverage)	Codex
SF110	2%	2% (line coverage), 1% (branch coverage)	Codex

II. Artificial Intelligence in Software Testing

Applications in the software testing life cycle (STLC) can be used to identify the areas in which AI approaches have shown to be helpful in software testing research and practice. In both the middle and final stages of the software testing lifecycle (STLC), LLMs have been utilized efficiently [2].Various LLMs have been utilized in this context, and each is having a different impact. For reference, Table 1 provides a list of these LLMs along with the corresponding results.

A. Automatic Test Case Generation

Every software system has a set of test cases, and as the system becomes more complicated, more test cases become necessary, increasing the time and effort needed for effective testing. Artificial intelligence (AI) technologies allow for the analysis of large volumes of data related to use patterns and applied data features during interactions to predict and develop a variety of dynamic test scenarios required for the highest level of confidence. An innovative solution that addresses these issues by automating the development of test cases is automatic test case generation.

One of the key areas where LLMs excel is in generating test cases automatically. Automating the generation of test cases could eliminate manual work, reducing test times and the cost of developing and maintaining software [3]. By analyzing software requirements and specifications, LLMs can construct test cases that explore various code paths and edge cases. This capability enhances test coverage and assists in identifying previously unnoticed bugs.

B. Test Suit Optimization

Traditional software testing often involves the creation of extensive test suites, leading to increased testing time and resource consumption. LLMs can optimize these test suites by removing redundant or less impactful test cases while ensuring

979-8-3503-1803-6/23/$31.00 ©2023 IEEE

the same level of coverage. This approach streamlines the testing process and reduces overall testing efforts.

Organizations can dramatically improve their test suits, making them more productive, cost-effective, and time-saving, by utilizing the power of LLMs. This enables quality assurance teams and software developers to concentrate on crucial areas for development, ensuring the delivery of the best software solutions to end customers. An exciting new era of smarter, more efficient testing procedures is about to arise with the incorporation of LLM-driven optimization into the software testing workflow.

C. Bug Localization Using LLMs

Finding the root cause of software bugs can be a time-consuming task. LLMs can aid in bug localization by analyzing error logs, stack traces, and user feedback to pinpoint the problematic sections of code. This helps developers to focus their efforts on resolving critical issues promptly. LLMs are capable of intelligently analyzing these many sources of data, enabling the very precise detection of faulty code parts.

Developers can quickly identify the bug's core cause because of this analytical skill, which also helps them save vital time and resources. Additionally, LLMs' capacity for continual learning enables them to evolve and advance over time, becoming increasingly more proficient in detecting recurrent patterns and typical risks in the software source. They are therefore essential assets for long-term bug management and program maintenance.

III. ChatGPT For Testing

ChatGPT created by OpenAI, is trained on an extensive set of text produced by human-generated and is built on the GPT (Generative Pre-training Transformer) architecture. The most powerful and efficient model in the GPT-3.5 series is the GPT-3.5-Turbo, which has been designed for chat but also performs successfully for more conventional completing tasks [4]. GPT-3.5 Turbo was created using a process known as Reinforcement Learning from Human Feedback (RLHF).

In unit testing, AI can apply techniques like classification, regression, clustering, and dimensionality reduction on data for software testing goals. To produce more accurate and promising analyses and predictions, AI can combine many algorithms [1]. ChatGPT can analyze given requirements and generate relevant test data and test cases based on the context and the desired outcome for a faster and more efficient testing process, and also it can be used to generate more realistic and complex test cases that can help identify a wider range of software bugs and problems [5].

In order for the tests to be generated properly, analyzing the terms and phrases used in tools for testing is an important factor. Context plays an important role in language models to generate reliable results. As a result, examining the LM's prompts and descriptions is critical. Ambiguous or poorly created prompts can lead to incorrect test cases, preventing the aim of automation. For ChatGPT to generate contextually appropriate replies, it is vital to incorporate relevant context and domain-specific data.

```
public static async Task<string> ChatGptHttpCallAsync(string query)
{
    string response = string.Empty;
    using (var client = new HttpClient())
    {
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", _settings.APIKey);
        var payload = new { model = _settings.Model, prompt = query, max_tokens = _settings.MaxTokens, temperature = _settings.Temperature };
        var content = JsonConvert.SerializeObject(payload);
        var httpResponse = await client.PostAsync(_url, new StringContent(content, Encoding.UTF8, "application/json")).ConfigureAwait(false);

        if (httpResponse.IsSuccessStatusCode)
        {
            var result = await httpResponse.Content.ReadAsStringAsync().ConfigureAwait(false);
            var chatGptResponse = JsonConvert.DeserializeObject<ChatGptGetResponseModel>(result);
            var choices = chatGptResponse.Choices.Select(p => p.Text);
            response = string.Join(" ", choices);
        }
        else
        {
            Console.WriteLine($"Error: {httpResponse.StatusCode}");
        }
    }

    return response;
}
```

Fig. 1: Usage of ChatGPT for Testing

Assessing the **correctness** of unit tests generated by ChatGPT is essential. In a study, all 207 Java classes could be successfully generated by ChatGPT with GPT-3 without encountering any issues. 144 (69.6%) of these test cases could be correctly compiled and executed without the help of additional human effort [6].

Code coverage is another significant factor in unit tests. In another study, for all projects using Evosuite which is a tool that automatically generates unit tests for Java software, the average statement coverage (SC) could reach 74.2%, the maximum SC could reach 77.4%, and the minimum SC could reach 70.6%. In comparison, the average SC for ChatGPT could reach 55.4% [6].

To ensure that human developers can easily maintain, understand, and adjust ChatGPT-generated code, it must be readable. This is essential when code generated by ChatGPT is going to be updated and altered over time by other developers or integrated into pre-existing codebases. According to a study, the majority of ChatGPT-generated tests are considered as having an acceptable readability and are sometimes even obtained as having a greater **readability** than manually written tests [7].

IV. Applying AI To Testing

Creating tests for projects using AI is our goal. We aim to use ChatGPT to write unit tests of the projects. We want to accelerate the creation and validation of unit tests using the capabilities of this language model, improving the general effectiveness and reliability of our projects. We will be able to communicate test scenarios and specifications with ease because of ChatGPT's natural language features, enabling us to develop thorough and efficient test suites. With this strategy, we intend to shorten the software development cycle and provide high-quality software.

A. Implementation

In order to include ChatGPT in a project, an exploratory Proof of Concept (POC) was performed. The goal of this study was to highlight important areas that required attention for successful implementation. In the POC, requests were made using both the HTTP client and the OpenAI-DotNet package.

```
public const string ControllerUnitTestQuery = "Can you please, create a class of
   the code below with unit tests written with xunit and mock package in
  accordance with positive and negative cases? set the namespace such as
  CompanyName.ProjectName.WebAPI.Tests.Controllers, if it returns success you
  can use the following logic {serviceResponseHelperSuccessMockExample} else
  {serviceResponseHelperUnsuccessMockExample}  just T type needs to be dynamic.
  {codeBlock}";
```

Fig. 2: Prompt for Unit Test Generation

During the process, it was discovered that certain parts of the.NET package would benefit from additional development. As a result, it was decided to focus on utilizing the HTTP client to perform the actual implementation because it provides greater flexibility for future development and improvements.

We integrated ChatGPT API into the project with a helper class. The provided code in Fig. 1 is a C# method that makes an asynchronous call to the ChatGPT API in order to interact with the language model and obtain an answer based on the specified query. The only variable the method accepts as an argument is the input text or prompt that will be used to communicate with the ChatGPT model. A string variable named is initialized inside the procedure to hold the eventual response obtained from the ChatGPT API. An API connection is made by using the HttpClient class. The method provides the relevant HTTP request headers, such as the authorization token needed to access the API.

Prompts are created to get useful and correct working responses according to the details of the structures such as libraries, frameworks used, and maintaining code standards. As shown in Fig. 2, by modifying the input to our project's specific structures, libraries, frameworks, rules, and coding standards, we set the foundation for meaningful and accurate outputs. This methodical approach improves the project's overall quality while also streamlining the development process.

Another important point for accurate responses is the temperature value. the temperature value plays a key role in getting the desired result. The temperature accepts values between 0 and 1. Higher temperatures (closer to 1) provide more innovative and diverse outcomes when text is generated using a language model, whereas lower temperatures (closer to 0) produce more reliable results. We set the temperature to 0.5 for unit test production to get more precise and effective replies. To find the ideal balance between originality and consistency depending on particular use cases, it is crucial to experiment with various temperature settings.

B. Understanding the Impacts

The use of ChatGPT for AI-driven test generation has proven to be a significant endeavor with major impacts on software quality and the advancement of the project. Our main goal has been to construct tests for projects with efficiency and effectiveness by using the strength of this language model. We have been able to build unit tests quickly with the integration of ChatGPT into our test generation process, considerably speeding up the entire development cycle. By providing thorough test coverage, this acceleration not only saves critical time but also improves the reliability and adaptability of our projects.

The seamless communication of test scenarios and requirements is one of this study's most important impacts. The natural language features of ChatGPT make it possible to clarify testing requirements clearly and concisely, promoting productive cooperation between developers and testers. Because of the faster communication process, test suites are created with higher precision, relevancy, and comprehensiveness. As a consequence, the software's quality is improved, lowering the possibility that the finished output would contain faults and mistakes.

Furthermore, the validation stage of our projects has been completely transformed by AI-powered test creation. ChatGPT enables our team to concentrate on more difficult and crucial parts of software development, such as design, architecture, and user experience, by automating the production of unit tests. This increases our developers' efficiency while also enabling us to provide higher-quality software in less time.

The use of ChatGPT to perform AI-driven test generation has produced noticeable advantages, removing testing bottlenecks and enhancing the consistency, effectiveness, and quality of software. We consider by implementing this strategy, the field of software development will be shaped in the future, opening opportunities for creative and effective solutions.

V. Challenges and Limitations

Despite the promising potential of LLMs in software testing, some challenges and limitations must be addressed. Ethical considerations, data privacy, and bias in training data are critical concerns that require thorough attention. Additionally, LLMs can face difficulties in comprehending code-specific nuances, hindering their accuracy in certain scenarios.

Due to the need for access to large volumes of data for training, there is a chance that LLMs can unintentionally expose sensitive data or violate privacy laws. Strong data anonymization and access control techniques must be used by developers and researchers to reduce this risk. Also, they might have any biases in the data, which would result in unfair or biased results. To address this, procedures for bias identification and data pretreatment should be used to guarantee fairness and objectivity in the process. On the other hand, LLMs can have difficulty understanding the nuances of a given code, which may reduce their accuracy in some circumstances. When working with complicated codebases or highly specialized fields, where particulars are vital to the functionality of the software, this constraint becomes more obvious. To better the models' comprehension of code semantics and context, this problem requires continual research and improvement.

A multidisciplinary strategy is required to overcome these obstacles and understand the potential advantages of LLMs in software testing. An appropriate compromise between innovation and ethical LLM usage can be achieved by collaboration between software developers, ethicists, and subject matter experts.

As LLM technology continues to evolve, future research in AI-based software testing should focus on mitigating the identified challenges. Developing LLMs specialized in code

comprehension, addressing bias issues, and enhancing their interpretability will further strengthen their applicability in testing methodologies.

To avoid the models from reproducing or reinforcing unfair or discriminatory outcomes, this can be accomplished by using diverse and inclusive training data as well as bias detection and correction approaches. Additionally, improving LLMs' interpretability is essential. It can be difficult to comprehend these models' decision-making processes since they are black boxes. Researchers should concentrate on creating comprehensible AI methods for LLMs to remedy this. It will be simpler for developers and testers to trust and successfully comprehend the outcome if methodologies are developed to give insights into how LLMs arrive at particular test results.

VI. Conclusion

In conclusion, recent technologies like ChatGPT and the development of artificial intelligence (AI) have created new opportunities in a variety of industries, including software development and testing. ChatGPT serves as a reference to the major developments made in this field as a result of the quick growth of AI. As KoçDigital we used LLM for software testing procedure.

AI-powered software testing, specifically leveraging LLMs, holds tremendous potential in revolutionizing traditional testing methodologies. The integration of LLMs in test case generation, test suite optimization, and bug localization can lead to more efficient and reliable software development practices. However, careful attention to ethical concerns and ongoing research is vital to fully harness the capabilities of LLMs in software testing and to realize their true benefits.

References

[1] Hussam Hourani, Ahmad Hammad, and Mohammad Lafi. The impact of artificial intelligence on software testing. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pages 565–570. IEEE, 2019.

[2] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software testing with large language model: Survey, landscape, and vision. *arXiv preprint arXiv:2307.07221*, 2023.

[3] Arjinder Singh and Sumit Sharma. Automated generation of functional test cases and use case diagram using srs analysis. *International Journal of Computer Applications*, 124(5), 2015.

[4] OpenAI. Models. [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5/.

[5] CodiumAI. ChatGPT for Automated Testing: Examples and Best Practices. [Online]. Available: https://www.codium.ai/blog/chatgpt-for-automated-testing-examples-and-best-practices/.

[6] Yutian Tang, Zhijie Liu, Zhichao Zhou, and Xiapu Luo. Chatgpt vs sbst: A comparative assessment of unit test suite generation. *arXiv preprint arXiv:2307.00588*, 2023.

[7] Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng. No more manual tests? evaluating and improving chatgpt for unit test generation. *arXiv preprint arXiv:2305.04207*, 2023.