«Системні технології» 2 (151) 2024 «System technologies»
DOI 10.34185/1562-9945-2-151-2024-10
UDC 004.4:004.8

V.V. Hnatushenko, I.V. Pavlenko

THE USE OF GENERATIVE ARTIFICIAL INTELLIGENCE IN SOFTWARE TESTING

Abstract. This article explores the potential of using generative artificial intelligence (AI) for software testing, reflecting on both the advantages and potential drawbacks of this emerging technology. Considering the vital role of rigorous testing in software production, the authors ponder whether generative AI could make the testing process more efficient and comprehensive, without the need to increase resources. The article delves into the current limitations of this technology, emphasizing the need for continuous exploration and adaptation. It concludes with a summation of potential innovative solutions and avenues for future investigation. The paper encourages discussions surrounding the question of fully automated testing and the role of hu man specialists in the future of QA. It ultimately provides a thought provoking reflection on the intersection of emerging technologies, and their societal impacts.
Keywords: generative AI, GenAI, software testing, large language model, test automation, AI augmented testing, AI assistant, autonomous test creation

Statement of the problem. In the previous few years there was a significant development in the pre-trained generative AI based on large language models (LLMs) with transformers and the services built upon them, OpenAI's ChatGPT, Google's Gemini, Anthropic's Claude, GitHub Copilot, Amazon CodeWhisperer and others. However, there remain several challenges and uncertainties regarding the effective integration of generative AI into software testing frameworks. This includes issues such as ensuring the reliability and accuracy of generated test cases, adapting AI algorithms to diverse software environments, and addressing ethical concerns surrounding autonomous testing systems. The following questions remain unanswered. At this stage of their development, what can GenAI on LLMs with transformers bring to software testing, and what can be expected from their further advancement? For which software testing tasks can GenAI on LLMs with transformers be used?

Analysis of recent studies and publications. Considering that GenAI on LLMs with transformers is an emerging topic [1–3], the number of scientific articles on the use of this type of GenAI in software testing is relatively small compared to

more developed topics in the field of information systems. In the work [3] the authors note that application of AI/ML has a long history in software engineering (SE) research. However, the use of GenAI specifically, is a more emerging topic. While the promise of GenAI has been acknowledged for some time, progress in the research area has been rapid. GenAI was not a prominent research area in SE until 2020. Following the recent improvements in the performance of these systems, especially the release of services such as GitHub Copilot and ChatGPT-3, research interest has now surged across disciplines, including SE. At the present stage of development of the considered models, there is a relatively rapid (a couple of times a year) growth in the number of model parameters, the volume of data used for their training, and the size of the model context. This leads to rapid growth in the capabilities of the models. Thus, this topic can be characterized as fast-changing. In connection with this, in a work [3] devoted to the analysis of existing publications for mid-2023, it is noted that: "A comprehensive and systematic review might not be suitable for research on GenAI in software development at the time this research being conducted". The authors further justify the use of preprints and other non-peer-reviewed works: "Firstly, research work is rapidly conducted and published on the topic, hence rendering the findings of a comprehensive review probably outdated shortly after publication. Secondly, we found a lot of relevant work as gray literature from non-traditional sources such as preprints, technical reports, and online forums. These sources may not be as rigorously reviewed or validated as peer-reviewed academic papers, making it difficult to assess their quality and reliability. Thirdly, we would like to publish the agenda as soon as possible to provide a reference for future research. A systematic literature review would consume extensive effort and time, which might be obsolete by the time the review is complete". They suggest conducting a literature review as follows: "Our strategy is to conduct focused, periodic reviews to capture the most current and relevant information without the extensive resource commitment of a comprehensive review. This approach allows for agility in keeping up with the latest developments without claiming comprehensiveness and repeatability" [3]. In this paper, in addition to the literature review, focus group surveys are conducted: "We conducted four structured working sections as focus groups to identify, refine, and prioritize Research Questions on GenAI for SE… …All participants are SE researchers who have experience or interest in the topic…". It is noted: "Almost all of the existing studies on the topic (e.g. "[13][14][15]") are mostly experimental studies and thus do not take into consideration the industrial context. Therefore, how GenAI models deal with real-world software quality issues remains a mystery. We need more real-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

world case studies and industrial examples to understand the effectiveness of various tasks of SQA with GenAI. Moreover, it would be interesting to study how well GenAI enhances the productivity of SQA professionals" [3]. Because of what was stated above and the practical nature of the selected theme (the topic is mostly practical, so focus groups from industry specialists would be much better), I used meetings and webcasts of testing industry professionals as a kind of focus groups to explore ideas for potential opportunities and challenges when adopting GenAI in software testing activities [10 - 12].

Purpose of the study. The purpose of this study is to explore the capability of GenAI integrated with transformers within Large Language Models to tackle distinct challenges encountered in software testing, such as test case generation, bug prediction, and test data synthesis. Through an examination of existing literature and insights from domain experts regarding the implementation of GenAI with transformers in software testing contexts, this research aims to formulate pertinent inquiries. These critical questions will serve as a framework for discussions concerning the contemporary role of advanced Generative AI in augmenting and refining software testing methodologies.

Statement of the main research material. As an introduction in [4] authors state that software testing, an integral part of the development cycle. However, automated software testing can be challenging, and necessitates a high level of technical acumen. There is significant expertise required to appropriately test software, as evidenced by the existence of test engineers/architects. Furthermore, software testing and the writing of software tests can be repetitive, as Hass et al. note [16]. It's important to acknowledge the crucial role of testing in software production. Testing isn't just vital for ensuring software quality; it also plays a significant role in refactoring or transitioning between different project types (proof of concept, MVP, etc.). Having test coverage is especially crucial when working on projects using an Agile approach. During refactoring and migrations, it's essential to verify that functionality and data remain intact throughout the process. In the case of software, testing serves as a similar verification tool. To ensure reliable verification, having a sufficient level of test coverage is necessary. Insufficient test coverage can lead to limitations in the ability to make changes, consequently affecting project development. It might also demand extensive and comprehensive manual testing, thereby slowing down project progress. Creating adequate test coverage for a project, especially maintaining the functionality of tests while the project undergoes continuous changes during its active development phase, is a labor-intensive task.

Software testing plays a crucial role. Despite this, many projects either aren't tested or are tested inadequately. This is attributable to the inherent complexity of testing itself. Approaching testing as "exhaustive," meant to cover all paths in the code or all possible input data, has highlighted that achieving complete software testing is impossible under these conditions. The sheer volume of potential input data, code execution paths, and system states is immense, rendering “exhaustive” testing theoretically impossible. As a result, in testing, it's necessary to establish a "stopping criterion," reaching which is regarded as a tradeoff between achieving as comprehensive testing as possible and the resources allocated for testing within the project's scope. However, the task of testing a project remains complex and unattainable in its entirety. There's hope that generative AI might assist in conducting more comprehensive testing without increasing the resources dedicated to testing.

Generative AI based on large language models with transformers is a new technology. We're only beginning to understand its capabilities and potential applications.

The primary tasks of testing can be generally described as follows:

1. Requirements analysis
2. Creating meaningful tests
3. Achieving code coverage targets
4. Maintaining the test suites
5. Documenting

When performing the aforementioned tasks, a human specialist encounters the following difficulties:

1. Requires a lot of effort
2. Requires a lot of time
3. Overwhelming for a person

Often, tests and test data created for testing are very similar from one test to another. During active project development and intensive changes to its codebase, a significant amount of effort and time are required to maintain the functionality of tests. Such work appears preferable for automation. However, automation based on predefined procedures does not yield the desired effect. It seems possible that generative AI based on LLM with transformers at this stage of development could help solve these tasks. It can be assumed that generative AI might assist in the following tasks:

- Requirements analysis.
- Generating test plans.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

• Generating test scenarios.

• Generating mocks.

• Generating test data.

• Extending and supplementing existing test sets.

• Maintaining the relevance of tests.

• Defect detection and prediction.

• Reducing maintenance cost by eliminating redundant tests.

• Bug reports writing.

• Generating documentation.

What advantages (capabilities) of applying generative AI can be expected:

• "Self-healing" of test suites when changes are made to the source code.

• Generating tests in the background or without using resources designated for development.

• Analysis of the user interface.

• Improving the productivity of human specialists.

• Generating tests from models or data description languages.

• Generating tests from informal descriptions, documentation written in natural languages, or audio descriptions.

• Optimizing test suites.

However, at this stage of generative AI development, attempting to use it brings some challenges:

• The field is rapidly evolving. Approaches and tools developed might become outdated before their completion with the publishing of more advanced models.

• High variability in generated tests (Deterministic vs. Probabilistic).

• Incorrect code generation.

• Inconsistent/Unreliable output.

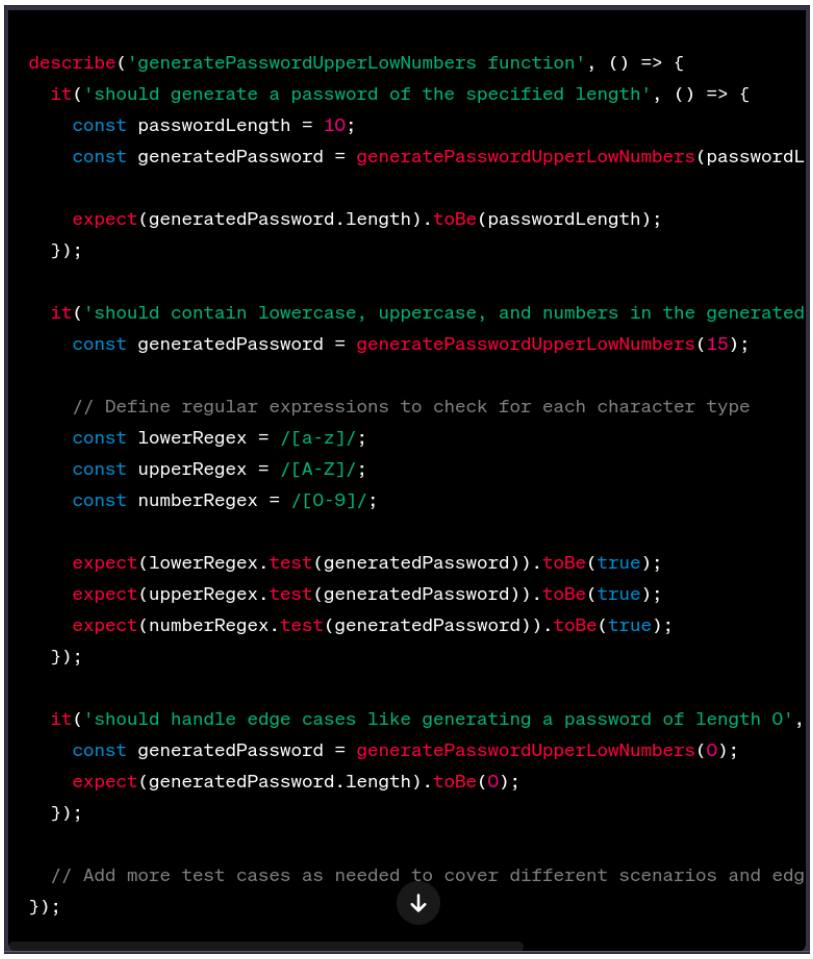
• Performance issues due to reliance on external systems.

• Costs associated with using generative AI services.

• Project code exposure when using external services, which might be unsuitable for certain cases.

One of the problems in generating tests using LLM is the probabilistic nature of content generation. While test cases and the source code of applications and tests themselves have a deterministic nature, LLMs built on transformers rely on a probabilistic model. This inherent probabilistic nature poses complexity in using this type of generative AI in tasks related to code generation. Additional efforts are required in post-processing the results either by a human or possibly even using the same

generative AI, not for generating but for analyzing the outcome, compiling, or checking for functionality. There might be a need for an additional step in code generation, asking the generative AI model to correct identified errors or mismatches. This process demands the development of additional criteria for the generated code to meet the given task. It involves work on selecting prompts for the model (prompt engineering), constructing a multi-step content generation system. However, due to the probabilistic model [5], possible model hallucinations [6], obtaining the expected result remains unguaranteed and requires verification by a human. You can see an example below where Figure 1 and Figure 2 contain different tests generated by ChatGPT 3.5 from the same prompt.



```
describe('generatePasswordUpperLowNumbers function', () => {
  it('should generate a password of the specified length', () => {
    const passwordLength = 10;
    const generatedPassword = generatePasswordUpperLowNumbers(passwordL

    expect(generatedPassword.length).toBe(passwordLength);
  });

  it('should contain lowercase, uppercase, and numbers in the generated
    const generatedPassword = generatePasswordUpperLowNumbers(15);

    // Define regular expressions to check for each character type
    const lowerRegex = /[a-z]/;
    const upperRegex = /[A-Z]/;
    const numberRegex = /[0-9]/;

    expect(lowerRegex.test(generatedPassword)).toBe(true);
    expect(upperRegex.test(generatedPassword)).toBe(true);
    expect(numberRegex.test(generatedPassword)).toBe(true);
  });

  it('should handle edge cases like generating a password of length 0',
    const generatedPassword = generatePasswordUpperLowNumbers(0);
    expect(generatedPassword.length).toBe(0);
  });

  // Add more test cases as needed to cover different scenarios and edg
});
```

Figure 1 - Case 01

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

```
describe('generatePasswordUpperLowNumbers function', () => {
  it('should generate a password of the specified length', () => {
    const passwordLength = 10;
    const generatedPassword = generatePasswordUpperLowNumbers(passwordL

    expect(generatedPassword.length).toBe(passwordLength);
  });

  it('should generate a non-empty password by default', () => {
    const generatedPassword = generatePasswordUpperLowNumbers();

    expect(generatedPassword.length).toBeGreaterThan(0);
  });

  it('should only contain characters from the defined character pools',
    const generatedPassword = generatePasswordUpperLowNumbers(20);

    const lowercase = 'abcdefghijklmnopqrstuvwxyz';
    const uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    const numbers = '0123456789';
    const pool = lowercase + uppercase + numbers;

    for (let i = 0; i < generatedPassword.length; i++) {
      expect(pool.includes(generatedPassword[i])).toBe(true);
    }
  });

  it('should return an empty string for zero password length', () => {
    const generatedPassword = generatePasswordUpperLowNumbers(0);
```

Figure 2 - Case 02

The paper [2], however, notes that software testing appears to be in a better position regarding the impact of inaccuracies in the generated test code, compared to business logic code generation: “However, there have been concerns about the correctness and reliability of the code generated by LLMs, as some studies have shown that the code generated by LLMs may not always be correct, or may not meet the expected software requirements. By comparison, when LLMs are used for software testing tasks, such as generating test cases or validating the correctness of software behavior, the impact of this problem is relatively weaker. This is because the primary goal of software testing is to identify issues or problems in the software system, rather than to generate correct code or meet specific software requirements. At worst, the only consequence is that the corresponding defects are not discovered. Furthermore, in some cases, the seemingly incorrect outputs from LLMs may be beneficial for testing corner cases in software and can help uncover defects. Taken in this sense, we think the LLMs are a natural match with software testing”.

Thus, the question arises whether, at this stage of generative AI development, it is possible to achieve automatic generation of a sufficiently complete set of tests and whether it should be pursued. Or should the current goal be to build a generative AI – human system using AI as an assistant? Some projects declare and attempt to move towards automated testing. For example, projects like Pythagora [7], TestCraftApp [8]. It can be expected that advancing the use of generative AI for testing will enhance test coverage, improve the cost-effectiveness ratio of testing, and expand the implementation of testing practices among companies and projects.

At the current level of development, automated creation of acceptable, not to mention fully covered, tests are rather impossible. It seems more appropriate to aim for test creation in cooperation with a human (developer) in assistant mode. Attempting complete automation might result in significant resource expenditure without achieving an acceptable outcome. It seems reasonable to start with a human-driven AI assistant, while further continuing to create a framework for as complete and comprehensive test automation as possible. Such paid platforms exist, created even before the use of AI on LLM.

At present, it can be stated that generative AI does not replace a tester but rather serves as a force multiplier, a "new electricity," as noted by the renowned machine learning expert Andrew Ng [9].

Efforts should also be directed towards creating a bridge (synergy) between AI and the practices and knowledge already established by humans (human specialists) in QA. Generative AI may help achieve greater efficiency in testing, but human involvement and knowledge remain critical for defining testing objectives and evaluating results.

It's interesting to note that many anticipated that the emergence of AI would allow humans to engage in creativity by eliminating routine tasks. However, it appears that, on the contrary, LLMs handle creative tasks better than tasks requiring attention to details and precision. It's possible that even at the highest available level of generative AI development, we might end up with an analog of a human specialist with inherent issues in human-to-human interaction, such as precise task setting (already emerging in prompt engineering and might remain unsolved in the future), understanding the task and its execution process by the performer in their own way. There might be incomplete or low-quality task execution (reports are already emerging that existing models gradually provide less comprehensive answers over time, indicating a sort of laziness in task execution).

Below are emerging questions that seem worth exploring.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Utilizing for context and processing not only the source code of a program but also existing documentation, both human-readable and formal documentation, DSLs (e.g., OpenAI) or formal service descriptions (e.g., protobuf). It might even involve the preliminary generation of documentation using AI as an additional supporting step. Perhaps an additional source in some generalized, processed and less formal form than the application source code, which the documentation is, will be useful in the current implementation of generative AI (LLM with transformers) to achieve better results.

Prompt engineering.

Fine-tuning the model specifically as a testing specialist.

How to best feed source code into the model?

Is it preferable to use and develop models with larger prompt token limits? One significant issue with using a large prompt size is that the model may forget parts of the prompt.

Alternatively, using retrieval augmented generation (RAG), or perhaps models fine-tuning on project artifacts, might yield a more suitable result.

Another practical question regarding generative AI is whether to lean towards more powerful proprietary models or use even smaller open-source models, possibly with fine-tuning or RAG, to achieve the desired outcome.

Findings. This article explores the potential of using generative AI for software testing, reflecting on both the advantages and potential drawbacks of this emerging technology. To provide the above analysis a review of existing publications was conducted. For initial assessments of interactions with LLM, the ChatGPT version 3.5 service and API access to the GPT 3.5 and GPT 4 models were used.

REFERENCES

1. Fan Angela, Gokkaya Beliz, Harman Mark, Lyubarskiy Mitya, Sengupta Shubho, Yoo Shin, Zhang Jie M. Large Language Models for Software Engineering: Survey and Open Problems // *arXiv. 2023. DOI:* https://doi.org/10.48550/arXiv.2310.03533.

2. Wang Junjie, Huang Yuchao, Chen Chunyang, Liu Zhe, Wang Song, Wang Qing. Software Testing with Large Language Model: Survey, Landscape, and Vision // arXiv. 2023. DOI: https://doi.org/10.48550/arXiv.2307.07221.

3. Nguyen-Duc Anh, Cabrero-Daniel Beatriz et al. Generative Artificial Intelligence for Software Engineering – A Research Agenda // arXiv. 2023. DOI: https://doi.org/10.48550/arXiv.2310.18648.

4. Feldt Robert, Kang Sungmin, Yoon Juyeon, Yoo Shin. Towards Autonomous Testing Agents via Conversational Large Language Models // arXiv. 2023. DOI: https://doi.org/10.48550/arXiv.2306.05152.
5. Keen M. How Large Language Models Work // IBM Technology. 2023. URL: https://www.youtube.com/watch?v=5sLYAQS9sWQ.
6. Keen M. Why Large Language Models Hallucinate // IBM Technology. 2023. URL: https://www.youtube.com/watch?v=cfqtFvWOfg0.
7. Pythagora project Github repository // Pythagora. 2023. URL: https://github.com/Pythagora-io/pythagora.
8. TestCraftApp project Github repository // TestCraft. 2023. URL: https://github.com/TestCraft-App/test-craft-api-v1.
9. Ng Andrew. The Near Future of AI // Stanford eCorner. 2023. URL: https://www.youtube.com/watch?v=KDBq0GqKpqA.
10. Kirilenko Igor, Jakubiak Nathan. Pros & Cons of Generative AI in Software Testing // Parasoft. 2023. URL: https://www.youtube.com/watch?v=57HTehOnll0.
11. Kirilenko Igor, Jakubiak Nathan. Use AI to Achieve Continuous Software Quality // Parasoft. 2023. URL: https://www.youtube.com/watch?v=mUPL7qJFtKA.
12. Kirilenko Igor, Jakubiak Nathan. Generative AI: the future of testing? - Test Smarter, Not Harder Event // Magnifai. 2023.
URL: https://www.youtube.com/watch?v=9Hfb4nW4uSg.
13. Liu H., Liu L., Yue C., Wang Y., Deng B. Autotestgpt: A system for the automated generation of software test cases based on chatgpt. 2023.
URL: https://ssrn.com/abstract=4584792.
14. Yuan Z., Lou Y., Liu M., Ding S., Wang K., Chen Y., Peng X. No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation. 2023 // arXiv. DOI: https://doi.org/10.48550/arXiv.2305.04207.
15. Ma W., Liu S., Wang W., Hu Q., Liu Y., Zhang C., Nie L., Liu Y. The scope of chatgpt in software engineering: A thorough investigation. 2023 // arXiv *preprint arXiv:2305.12138*.
16. Haas R., Elsner D., Juergens E., Pretschner A., Apel S. How can manual testing processes be optimized? developer survey, optimization guidelines, and case studies // 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE 2021, Athens, Greece: Association for Computing Machinery, 2021, P. 1281–1291.

Received 14.03.2024.
Accepted 18.03.2024.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Використання генеративного штучного інтелекту в тестуванні програмного забезпечення

У роботі досліджується потенціал використання генеративного штучного інтелекту (GenAI) на основі великих мовних моделей (LLM) з трансформерами для покращення різних аспектів тестування програмного забезпечення. Акцент робиться на можливих практичних застосуваннях і проблемах, що виникають у цих нових підходах. Визначено проблеми тестування і потенціал генеративного ШІ для можливого їх вирішення або зниження їх впливу на ведення проектів програмних систем. Хоча генеративний ШІ на даному етапу розвитку не є повною заміною тестувальникам-людям, він пропонує значні перспективи як потужний допоміжний інструмент, який може трансформувати практики тестування. Очікуваними перевагами є "самовиліковні" тести, які адаптуються до змін коду; генерація тестів у фоновому режимі; можливість генерувати тести з різних джерел (моделей, описів природною мовою, неофіційної документації), і в решті-решт – підвищення продуктивності праці тестувальників. Зазначено виклики використання генеративного ШІ на великих мовних моделях з трансформерами.

Гнатушенко Володимир Володимирович - д.т.н., професор, завідувач кафедри інформаційних технологій та комп'ютерної інженерії Національного технічного університету «Дніпровська політехніка» (м. Дніпро).

Павленко Єгор Вікторович - аспірант кафедри інформаційних технологій та комп'ютерної інженерії Національного технічного університету «Дніпровська політехніка» (м. Дніпро).

Hnatushenko Volodymyr - Doctor of Technical Sciences (Dr.-Ing. habil.), Professor, Head of Department of Information Technology and Computer Engineering, Dnipro University of Technology, Dnipro, Ukraine.

Pavlenko Iegor - doctoral student of Department of Information Technology and Computer Engineering, Dnipro University of Technology, Dnipro, Ukraine.