Büyük Dil Modelleri ile Elektrikli Araç Yazılım Test Senaryolarının Otomatik Üretimi ve Değerlendirmesi

Automated Generation and Evaluation of Electrical Vehicles Software Test Scenarios Using Large Language Models

Cem BAĞLUM[1], *Uğur YAYAN*[2], *Ahmet YAZICI*[3]

[1]Bilgisayar Mühendisliği Bölümü
Eskişehir Osmangazi Üniversitesi, Eskişehir
cembglm@gmail.com

[2]Yazılım Mühendisliği Bölümü
Eskişehir Osmangazi Üniversitesi, Eskişehir
ugur.yayan@ogu.edu.tr

[3]Bilgisayar Mühendisliği Bölümü
Eskişehir Osmangazi Üniversitesi, Eskişehir
ayazici@ogu.edu.tr

Özetçe

Elektrikli araçlarda akıllı enerji yönetim sistemleri büyük önem taşır; bu sistemlerin yüksek oranda yazılım içermesi ve yazılımların gürbüzlüklerinin emniyet açısından kritik olması nedeniyle yoğun bir şekilde test edilmeleri gerekmektedir. Bu nedenle, yazılımların gürbüzlüklerin test edilmesi için test senaryoları üretilmesi gerekmektedir. Bu çalışma, Google'ın CodeGemma ve Meta'nın CodeLLaMa büyük dil modellerinin elektrikli araç yazılımları için test senaryoları üretme yeteneklerini incelemektedir. CodeGemma, kod tamamlama ve doğal dil anlama gibi görevlerde üstün performans gösterirken, CodeLLaMa kod üretimi ve hata tespitinde uzmanlaşmıştır. Her iki model, elektrikli araç yazılımlarına ait aynı kaynak kodlarla maksimum sayıda test senaryosu üretmiş ve performansları dil bilgisi ve kaynak kod uyumluluğu açısından çapraz değerlendirme ile incelenmiştir. Sonuçlar, GPT-4o ve LLaMa3 modelleri ile puanlanarak analiz edilmiştir. Çalışmanın amacı, test senaryolarının kalitesini artırarak elektrikli araçlarda bulunan akıllı enerji yönetim sistemlerindeki yazılım test süreçlerini iyileştirmektir. Bulgular, dil modellerinin, gerçekleştirilen test süreçlerinde verimliliği artırmada önemli bir rol oynayabileceğini göstermektedir.

Abstract

Intelligent energy management systems in electric vehicles are of great importance; these systems heavily rely on software, and the robustness of these software components is critical for safety, necessitating extensive testing. Therefore, it is essential to generate test scenarios to assess the robustness of the software. This study examines the capabilities of Google's CodeGemma and Meta's CodeLLaMa large language models in generating test scenarios for electric vehicle software. CodeGemma excels in tasks such as code completion and natural language understanding, while CodeLLaMa specializes in code generation and error detection. Both models generated the maximum number of test scenarios from the same source code of electric vehicle software, and their performance was evaluated through cross-assessment for linguistic accuracy and code compatibility. The results were analyzed using GPT-4o and LLaMa3 models. Test scenarios are created and evaluated using CodeGemma and CodeLLaMa. The aim of this study is to enhance the quality of test scenarios and improve the software testing processes in intelligent energy management systems in electric vehicles. The findings suggest that language models can play a significant role in increasing efficiency in the testing processes.

1. Introduction

Smart energy management systems in electric vehicles are highly important, and these systems incorporate a significant amount of software; due to the critical nature of the robustness of these software systems for safety, they must undergo rigorous testing. The automatic generation of test scenarios for electric vehicle software is crucial for enhancing software quality and accelerating development processes. Test scenarios are utilized to validate the expected behaviors of the software, detect potential errors, and increase the software's reliability. The use of large language models for generating test scenarios is shown in our study and the literature to have a positive impact on efficiency and effectiveness in software testing processes.

In this study, the test scenario generation capabilities of two different large language models, CodeGemma developed by Google and CodeLLaMa developed by Meta, for electric vehicle software are examined [1,2]. CodeGemma is built on Google's Gemma models and demonstrates superior performance in tasks such as code completion, natural language understanding, and mathematical reasoning [3]. CodeLLaMa, on the other hand, is built on Meta's LLaMa2, trained on extensive datasets, and specialized in areas like code generation, error detection, and optimization [4]. Both models were provided with the same source code and commands related to electric vehicle software and were asked to generate

as many test scenarios as possible. Their performance was evaluated in terms of linguistic quality and compatibility with the source code. The main goal of the study is to improve software testing processes by enhancing the quality of the test scenarios produced by these models. The findings of the study indicate that language models can play a significant role in generating test scenarios.

Previous studies have shown that the integration of artificial intelligence technologies into software testing processes offers significant advantages in terms of accelerating these processes and enhancing their efficiency. Jaber et al. investigated the automatic generation of test scenarios using artificial intelligence and machine learning [5]. Li et al. discussed the role of artificial intelligence in test automation and its contributions to software testing processes [6]. Large language models, in particular, stand out as effective tools for the automatic generation of software test scenarios. Yang et al. introduced Text2Reaction, a large language model-based framework that enables robots to respond to environmental changes [7]. Ionescu and Enescu explored how ChatGPT can be used in the processes of creating and evaluating online tests [8]. Studies on the capabilities of large language models to detect and correct errors in Python programs demonstrate their effectiveness. Wuisang et al. evaluated ChatGPT's performance in automatic error detection and correction using the QuixBugs benchmark set [9].

Studies have shown that large language models can be successfully used in tasks that are more verbal in nature, such as the generation of test scenarios. Schafer et al. presented a comparative analysis of large language model-based test generation techniques versus traditional methods [10]. Lee and Hsiang examined the use of the BERT model in patent classification tasks [11]. Wei et al. evaluated the performance of the DistilBERT model in reviewing legal documents [12]. Automatic test data generation is an important method for increasing code coverage in software. Avdeenko and Serdyukov proposed an approach aimed at generating test data using genetic algorithms [13]. Studies on the effectiveness of completing software test scenarios using control flow graphs play a significant role in increasing test coverage. Zhang et al. presented coverage measurement methods for equipment software system testing [14]. Additionally, Caglar et al. discussed how cloud-based platforms like ChArIoT can improve software testing processes [15]. ISTA has been developed as a tool supporting various coverage criteria for deep neural networks. Zheng et al. examined the impact of this tool on test case generation and optimization [16].

This study, which aims to enhance the efficiency and accuracy of artificial intelligence techniques used in generating test scenarios, we go beyond existing methods in the generation and evaluation of test scenarios for electric vehicle software. It has been demonstrated that automatic test scenario generation using language models offers significant forward-looking findings in the fields of software engineering and test automation. A comprehensive methodology was followed to examine the effectiveness of using language models in generating and evaluating test scenarios for electric vehicle software. The test scenario generation capabilities of CodeGemma, developed by Google, and CodeLLaMa, developed by Meta, were compared for electric vehicle software. At the initial stage of the study, both models were provided with the same source codes and commands to generate the maximum number of test scenarios. The generated test scenarios were evaluated based on criteria such as linguistic accuracy, adherence to grammatical rules, and compatibility with the source code. During this process, the models performed a cross-check by analyzing each other's generated scenarios. Where necessary, additions were made to enhance the quality of the test scenarios, and in the final step, the test scenarios were scored using the GPT-4o and LLaMa3 models to evaluate their overall performance. The findings revealed that both CodeGemma and CodeLLaMa models achieved high accuracy and compatibility in generating test scenarios. However, CodeGemma generally exhibited superior performance. It was determined that both models have the potential to improve the testing processes of electric vehicle software. These results indicate that large language models can make significant contributions to the fields of software engineering and test automation, playing a critical role in increasing efficiency and effectiveness beyond existing methods.

In the subsequent sections of the study, the details of the CodeGemma and CodeLLaMa models, as well as the software and hardware components used, will be elaborated upon. The third section will detail the proposed method and the command engineering techniques used, explaining the procedures undertaken to generate the maximum number of test scenarios. Finally, in the fourth section, the test scenarios generated by the models will be compared in terms of linguistic and source code compatibility, and the results will be evaluated.

2. Large Language Models Used in the Study

The increasing importance of artificial intelligence technologies in software engineering processes necessitates the use of these technologies for the automatic generation of test scenarios for electric vehicle software. The capabilities of CodeGemma, CodeLLaMa, LLaMa3, and GPT-4o models in generating and evaluating test scenarios for electric vehicle software using large language models are examined in detail (Table 1).

Table 1 Details of Large Language Models Used in the Study

Features	CodeGemma	CodeLLaMa	GPT-4o	LLaMa3
Developer	Google	Meta	OpenAI	Meta
Training Data	500 billion tokens (code and English data)	Extensive code and data sets	Text, audio, and image data	Extensive languag e and code data
Core Capabilities	Code completion, code generation, natural language understandi ng, mathematic al reasoning	Code generation , error detection, optimizati on	Text, audio, and image processin g	Code generati on, test scenario generati on
Multilingua l Support	Python, JavaScript, Java, Kotlin,	Python, JavaScript , Java, C#,	Python, Java, C++, C#, JavaScrip	Python, Java, C++, C#,

	C++, C#, Rust, Go	C++, Go, Kotlin	t, Go, Rust, Swift, PHP, Ruby, HTML/CSS	JavaScript
Performance Evaluation	High accuracy and compatibility, suitable for electric vehicle software	Superior in API and microservice testing, compatible with electric vehicle software	High-speed and low-cost multimodal data processing, suitable for electric vehicle software	Extensive test scenario generation, optimized for electric vehicle software

2.1. CodeGemma

Developed by Google, CodeGemma is a high-performance model in the fields of code completion and generation, trained on 500 billion code tokens. Available in 2B and 7B parameter sizes, the model is effective in generating complex test scenarios thanks to its natural language processing and mathematical reasoning capabilities. CodeGemma, which produces strong and consistent outputs with the "Fill-in-the-Middle (FIM)" training approach, offers advantages in real-time software development projects with low latency. These capabilities accelerate testing processes, enabling the early detection of software bugs.

2.2. CodeLLaMa

Developed by Meta, CodeLLaMa is a large language model with extensive code and language comprehension capabilities. The model, which can generate complex test scenarios with its advanced natural language processing abilities, contributes to software development and test automation processes. CodeLLaMa can create test scenarios with high accuracy in various programming languages and software frameworks, making it suitable for API-based tests and microservices architectures. Developed with Python and PyTorch and optimized on AWS, the model operates efficiently on large datasets to produce real-time test scenarios. CodeLLaMa plays a crucial role in enhancing software quality and reducing error rates.

2.3. LLama3

LLama3 offers innovative artificial intelligence solutions in the field of software engineering and test automation [19]. The model excels in test scenario generation and coverage evaluation. With its extensive language understanding capacity and robust code generation capabilities, it creates test scenarios with high accuracy in various programming languages. Utilizing a vast amount of code and natural language data, LLama3 produces reliable test scenarios for complex software, prepares comprehensive scenarios for API and integration tests, and identifies missing components. These features make test processes completer and more reliable.

2.4. GPT-4o

Developed by OpenAI, GPT-4o is an exceptionally successful model in the field of natural language processing. Trained on a large dataset, this model can solve complex language and code problems. GPT-4o offers similar advantages to LLama3 in test scenario generation and coverage evaluation. It interprets user inputs to produce suitable test scenarios, thoroughly testing various use cases of the software. Additionally, it analyzes the coverage of the test scenarios it generates, identifying gaps to optimize the test processes and enhance software reliability.

In conclusion, the combined use of language models such as CodeGemma and CodeLLaMa, specifically tailored for the software domain, along with LLama3 and GPT-4o, has the potential to revolutionize software testing processes. CodeGemma and CodeLLaMa are employed to create and evaluate test scenarios, while LLama3 and GPT-4o are utilized to evaluate the outputs generated and cross-assessed by CodeGemma and CodeLLaMa in the final step. These models, with their superior performance in test scenario generation and coverage evaluation, significantly enhance software quality and reliability.

3. Method

We employed CodeGemma and CodeLLaMa to automatically generate and evaluate test scenarios for electric vehicles and smart energy management systems in the study. These models analyze the source code to identify potential and useful test scenarios, which are then cross-evaluated for linguistic suitability and source code compatibility. Finally, the GPT-4o and LLaMa3 models perform a thorough assessment, scoring the scenarios on a scale of 100 to compare the effectiveness of CodeGemma and CodeLLaMa.

CodeGemma is a model developed to identify potential and useful test scenarios based on the given source code. The model's operation process includes the analysis of the source code, the creation of test scenarios, and their evaluation. Both CodeGemma and CodeLLaMa follow similar working principles by analyzing the source code to identify all possible and beneficial test scenarios. These models conduct a comprehensive analysis by considering functions, conditions, and loops within the code. Test scenarios are generated using the "Brute Force Technique" and are produced step-by-step logically through the "Chain of Thought" structure. CodeLLaMa operates with similar processes but employs different algorithms and optimization techniques compared to CodeGemma, aiming to create more comprehensive and effective scenarios (Figure 1).

After the initial outputs, the test scenarios generated by CodeGemma and CodeLLaMa were cross-evaluated for linguistic suitability and source code compatibility using the prompt: "Evaluate the test scenarios generated by CodeGemma/CodeLLaMa. Analyze if they are linguistically correct, if the maximum test scenario number is reached, and if they are compatible with the source code. If the maximum number is not reached, add new test scenarios using the Brute Force Technique." The final evaluation by GPT-4o and LLaMa3 models used the prompt: "Evaluate the test scenarios and give a score out of 100 based on their linguistic correctness and compatibility with the source code like X%."

Using the generated and cross-evaluated test scenarios, the desired model can generate test code. In this approach, test code can be produced by employing a prompt that connects the generated test scenarios with the source code [20]. An example of such a prompt, which can be used to create test code corresponding to these scenarios (Table 2).

Table 2 Test Code Generation Prompt Example

Example Prompt to Create Test Code According to Test Scenarios
Generate test code based on the source code and evaluated test scenarios, ensuring that the code addresses potential security vulnerabilities, performance considerations, and adheres to best coding practices. The source code is: {source_code}. Evaluated Test Scenarios: {evaluated_test_scenarios}.

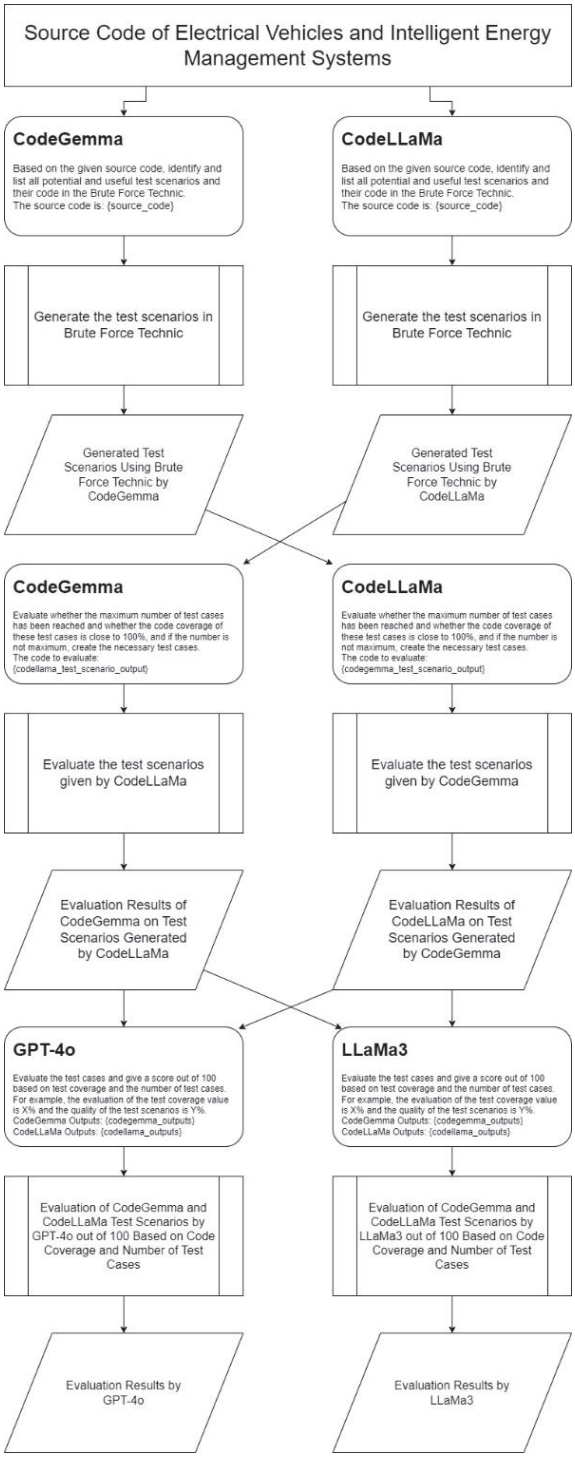

Figure 1 Structure Created Using Large Language Models

In the study, Python code was used to generate and evaluate test scenarios for electric vehicles and smart energy management systems [21]. The code produced route solutions based on a distance matrix using XML data and evaluated these solutions (Table 3).

The methods and evaluation processes used provided significant insights into the generation and optimization of automatic test scenarios. The integration of the "Chain of Thought" structure enabled more effective generation of test scenarios in logical steps, thereby increasing both the quality and quantity of the test scenarios.

4. Findings

The Python code used in the study was developed to automatically generate and evaluate test scenarios for electric vehicles and smart energy management systems. The code converts the ALNS solution provided as a string into a list structure and calculates the distances by finding each pair of elements in the distance matrix. This method is employed to optimize energy management and routing processes in electric vehicles. The pseudocode structure of the algorithm used is shown (Table 3).

Table 3 Pseudocode Structure of the Algorithm Used in the Study

1. Take XML data as input.
2. Extract the distance matrix from the XML data.
3. Extract the points from the XML data.
4. Initialize the variable string_solution.
5. Convert string_solution into route_lists using StringToList(string_solution).
6. Initialize the variable total_distance to 0.
7. For each route in route_lists:
8. For each item in the route:
9. Set the item as left_item.
10. Set the next item as right_item.
11. For each point in points:
12. If point.name equals left_item:
13. Set left_index to int(point.no) - 1.
14. If point.name equals right_item:
15. Set right_index to int(point.no) - 1.
16. End the loop.
17. Retrieve distance_matrix[left_index][right_index] and add it to total_distance.
18. End the loop.
19. End the loop.
20. Return total_distance.

The code takes a list of solutions and converts it into a Solution object, then uses the Brute Force technique to generate all possible solutions and calculate their costs. This study aims to automatically generate and evaluate test scenarios for electric vehicle software using Google's CodeGemma and Meta's CodeLLaMa models. Table 4 shows examples of the generated test scenarios. You can access the tests and codes created by the models on the study's GitHub page [20].

Table 4 Sample Test Scenarios

Model	**Created Sample Test Scenario**
CodeLLaMa	Testing different input files: The program takes an input file as an argument, which contains the information about the problem instance. Testing

	different input files can help identify any bugs or issues in the program.
CodeGemma	Valid Solution: Test if the given solution is a valid route configuration within the problem constraints.

The test scenarios generated by CodeGemma were evaluated with a linguistic accuracy rate of 100% and a source code compatibility of 95%. Similarly, the test scenarios produced by CodeLLaMa showed 100% linguistic accuracy and 90% source code compatibility. In the final evaluation by GPT-4o, it was noted that the test scenarios were 91% successful in terms of linguistic accuracy and source code compatibility. In the assessment by LLama3, the overall performance of the test scenarios was rated at 92%. Both models provided a wide range of scenarios, comprehensively testing different aspects of the software, but it was suggested that the inclusion of additional scenarios could offer more coverage.

The study compares the performance of test scenarios generated by Google's CodeGemma and Meta's CodeLLaMa models in terms of linguistic accuracy and source code compatibility. CodeGemma's linguistic accuracy rates are generally above 90%, with the highest performance being 100% in the "Valid Solution" scenario. The code compatibility rates for this model range from 85% to 95%. Particularly, in the "Valid Solution" and "Input Validation" scenarios, a 95% code compatibility rate was achieved. CodeGemma significantly enhanced the accuracy and validity of test scenarios by demonstrating high linguistic accuracy and source code compatibility overall (Table 5).

Table 5 Evaluation of the Created Test Scenarios

Test Scenario	CodeGemma Linguistic Accuracy (%)	CodeGemma Code Compatibility (%)	CodeLLaMa Linguistic Accuracy (%)	CodeLLaMa Code Compatibility (%)
Valid Solution	100	95	100	90
Route Length Validation	95	90	95	85
Customer Coverage	95	90	90	85
Charging Station Usage	90	85	85	80
Route Optimization	95	95	90	85
Solution Uniqueness	95	90	90	80
Error Handling	90	85	85	80
Performance Optimization	90	85	85	80
Input Validation	95	95	90	85
Coverage of All Customers	95	90	90	85

The linguistic accuracy rates of the CodeLLaMa model range from 85% to 100%, with the highest performance achieved at 100% in the "Valid Solution" scenario. The code compatibility rates of this model vary between 80% and 90%. Specifically, a 90% code compatibility was achieved in the "Valid Solution" and "Route Length Verification" scenarios (Figure 2).

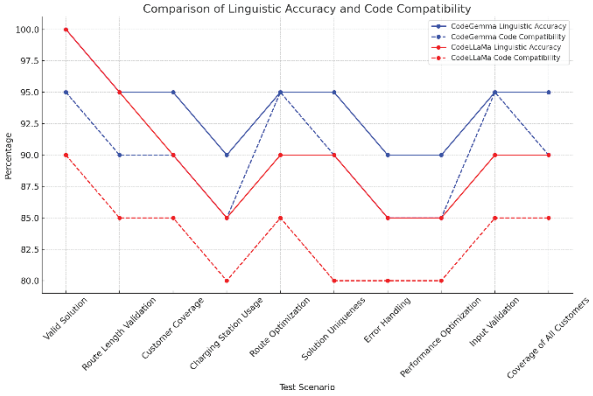

Figure 2 Performance Comparison of CodeGemma and CodeLLaMa Models

Although CodeLLaMa has provided satisfactory results in terms of linguistic accuracy and code compatibility, it has demonstrated lower performance compared to CodeGemma. These findings indicate that while CodeGemma achieves higher accuracy and compatibility in test scenario generation, both models have the potential to enhance the testing processes of electric vehicle software.

5. Conclusion

The complex and extensive codebases of electric vehicle software make manual test scenario generation time-consuming and costly. In our study, we examined how large language models accelerate this process and improve its quality, aiming to generate automatic test scenarios for electric vehicle software using Google's CodeGemma and Meta's CodeLLaMa models, and to evaluate the results by cross-checking them with OpenAI's GPT-4o and Meta's LLaMa3 models. The results demonstrate that by interpreting different codes in various ways, the CodeGemma and CodeLLaMa models enhance the quality and reliability of test scenarios with high accuracy and compatibility. This study shows that large language models improve software testing processes, providing significant time and cost savings by accelerating operations that are manually performed, thus contributing substantially to software development processes. Furthermore, due to the importance of software used in safety-critical electric vehicles and smart energy management, this study addresses the robustness of routing software in these systems by generating and verifying various test scenarios, highlighting the critical importance of testing such software. These findings confirm that large language models have vast potential in the fields of software engineering and test automation.

Acknowledgement

This work is supported by the Scientific and Technical Research Council of Turkey (TUBITAK), Contract No 222N269, project title: “OPEVA: Optimization of Electric Vehicle Autonomy”.

This paper is supported by the OPEVA project that has received funding within the Key Digital Technologies Joint Undertaking (KDT JU) from the European Union's Horizon Europe Programme and the National Authorities (France, Belgium, Czechia, Italy, Portugal, Turkey, Switzerland), under grant agreement 101097267.

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or KDT JU. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Google Developers, "CodeGemma Documentation," https://ai.google.dev/gemma/docs/codegemma. (2024).

[2] Meta, "Code Llama: A Large Language Model for Coding," https://ai.meta.com/blog/code-llama-large-language-model-coding/. (2024).

[3] Google, "Gemma Open Models," https://blog.google/technology/developers/gemma-open-models/. (2024).

[4] Meta, "LLaMA 2," https://llama.meta.com/llama2/. (2024).

[5] K. M. Jaber, Institute of Electrical and Electronics Engineers. Jordan Section, Institute of Electrical and Electronics Engineers. Region 8, and Institute of Electrical and Electronics Engineers, 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT) : proceedings : April 9-11, 2019, Le Royal Amman Hotel, Jordan.

[6] J. J. Li, A. Ulrich, X. Bai, and A. Bertolino, "Advances in test automation for software with special focus on artificial intelligence and machine learning," Software Quality Journal, vol. 28, no. 1. Springer, pp. 245–248, Mar. 01, 2020. doi: 10.1007/s11219-019-09472-3.

[7] Z. Yang et al., "Text2Reaction : Enabling Reactive Task Planning Using Large Language Models," IEEE Robot Autom Lett, May 2024, doi: 10.1109/LRA.2024.3371223.

[8] V. M. Ionescu and M. C. Enescu, "Using ChatGPT for Generating and Evaluating Online Tests," in 15th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2023 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/ECAI58194.2023.10193995.

[9] M. C. Wuisang, M. Kurniawan, K. A. Wira Santosa, A. Agung Santoso Gunawan, and K. E. Saputra, "An Evaluation of the Effectiveness of OpenAI's ChatGPT for Automated Python Program Bug Fixing using QuixBugs," in 2023 International Seminar on Application for Technology of Information and Communication: Smart Technology Based on Industry 4.0: A New Way of Recovery from Global Pandemic and Global Economic Crisis, iSemantic 2023, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 295–300. doi: 10.1109/iSemantic59612.2023.10295323.

[10] M. Schafer, S. Nadi, A. Eghbali, and F. Tip, "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation," IEEE Transactions on Software Engineering, vol. 50, no. 1, pp. 85–105, Jan. 2024, doi: 10.1109/TSE.2023.3334955.

[11] J. S. Lee and J. Hsiang, "Patent classification by fine-tuning BERT language model," World Patent Information, vol. 61, Jun. 2020, doi: 10.1016/j.wpi.2020.101965.

[12] F. Wei et al., "Empirical Study of LLM Fine-Tuning for Text Classification in Legal Document Review," in Proceedings - 2023 IEEE International Conference on Big Data, BigData 2023, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 2786–2792. doi: 10.1109/BigData59044.2023.10386911.

[13] T. Avdeenko and K. Serdyukov, "Automated test data generation based on a genetic algorithm with maximum code coverage and population diversity," Applied Sciences (Switzerland), vol. 11, no. 10, May 2021, doi: 10.3390/app11104673.

[14] J. Zhang, Y. Peng, and H. Yang, "A Test Design and Coverage Measurement Method for Equipment Software System Testing," in IEEE Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Institute of Electrical and Electronics Engineers Inc., 2022, pp. 720–730. doi: 10.1109/ITAIC54216.2022.9836534.

[15] O. Caglar, F. Taskin, C. Baglum, S. Asik, and U. Yayan, "Development of Cloud and Artificial Intelligence based Software Testing Platform (ChArIoT)," in 2023 Innovations in Intelligent Systems and Applications Conference, ASYU 2023, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/ASYU58738.2023.10296551.

[16] W. Zheng et al., "ISTA: Automatic Test Case Generation and Optimization for Intelligent Systems based on Coverage Analysis," in Proceedings - 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2023, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 758–762. doi: 10.1109/SANER56733.2023.00086.

[17] PyTorch, "PyTorch," https://pytorch.org/. (2024).

[18] Amazon Web Services, "Amazon Web Services," https://aws.amazon.com. (2024).

[19] Meta, "LLaMA 3," https://llama.meta.com/llama3/. (2024).

[20] GitHub, "EV-Software-Test-Scenarios-LLM", 2024. https://github.com/ESOGU-SRLAB/EV-Software-Test-Scenarios-LLM (2024)

[21] GitHub, "Array to Solution", 2024. https://github.com/ESOGU-SRLAB/EV-Software-Test-Scenarios-LLM/blob/main/src/test_file/array_to_solution.py.