Test Case Generation for Requirements in Natural Language - An LLM Comparison Study

Brahma Reddy Korraprolu
SERC, IIIT Hyderabad
Hyderabad, India
brahma.reddy@research.iiit.ac.in

Pavitra Pinninti
SERC, IIIT Hyderabad
Hyderabad, India
pavitra.pinninti@research.iiit.ac.in

Y. Raghu Reddy
SERC, IIIT Hyderabad
Hyderabad, India
raghu.reddy@iiit.ac.in

ABSTRACT

The rapid evolution of Large Language Models (LLMs) have opened new possibilities in automating tasks across the software developing life cycle, including test case generation This paper presents a comparative analysis of six LLMs in the context of generating test cases for technical requirements written in natural language (in this case English). We compare publicly available general purpose LLMs viz., BARD, ChatGPT3.5, Claude, Gemini, ChatGPT4.o (Omni) and Llama3. The generated test cases are tested against a Simulink model created for the corresponding set of requirements. The coverage metrics thus generated are used for a quantitative comparison of the LLMs.

CCS CONCEPTS

• **Software and its engineering → Software notations and tools**.

KEYWORDS

LLM, Large Language Models, Test Case Generation, Requirements

ACM Reference Format:
Brahma Reddy Korraprolu, Pavitra Pinninti, and Y. Raghu Reddy. 2025. Test Case Generation for Requirements in Natural Language - An LLM Comparison Study. In *Proceedings of Innovations in Software Engineering Conference (ISEC '25).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Large Language Models (LLMs) are Artificial Intelligence (AI) models trained on vast text data to understand and generate human-like language, typically using deep learning architectures like transformers. As all public LLMs are trained on a huge corpus of generic data without any domain specification it could be considered that an LLM is a general purpose response generator where the query could be from any domain in which the LLMs' training data has been sourced. Currently majority of LLMs are being used in tasks such as translation, content generation, summarizing, question and answers sentiment analysis etc. LLMs have achieved much progress in recent times in many domains and most of the popular LLMs are available in the form of apps or with simple user interfaces that assist with using the model as a service.

LLMs ,trained on a sufficiently large corpus of data, are anticipated to produce meaningful outcomes in various software development lifecycle tasks. The generation of test cases represents one such task that, while specific in its execution, can be largely domain-agnostic across numerous software applications. It is well established in design and development process that test cases can be generated from design models and/or code. For example, MATLAB test cases can be generated for the system models using Simulink Design Verifier[1] (SLDV). Present day practices are increasingly advocating a Shift-left approach, i.e., identifying and resolving bugs early in the development lifecycle to improve software quality, achieve better test coverage, get continuous feedback, etc. Automatic test case generation can speed up shift-left approach where in the test cases are generated at the same time as requirements.

The predominant format for software requirements specification documents is natural language. Unlike structured modeling languages, like UML and BMPL, there has been a notable lack of support for the generation of test cases from natural languages like English. LLMs trained on extensive data corpus have understanding of various code patterns, syntax, and semantics [4]. LLMs with this enhanced understanding can generate diverse & comprehensive test scenarios and test cases that effectively explore edge cases and reveal potential bugs. This capability becomes increasingly valuable in ensuring thorough and effective testing processes for complex systems. LLMs have the potential to create a wide array of test cases that encompass various testing scenarios, including Unit Testing, API Testing, Functional Testing, Integration Testing, Usability Testing, and Security Testing [2] In this paper, we compare auto test case generation capabilities of six publicly available LLMs namely: BARD, ChatGPT3.5, Claude, Gemini, ChatGPT4.o (Omni) and Llama3.

The main contribution of this paper is a quantitative analysis of the performance of publicly available LLMs with respect to their ability to produce test cases from human readable requirements. The paper address the following research questions:

(1) Can LLMs generate test cases for a software model using only its textual requirements as input?
(2) In what cases can LLMs not generate test cases from requirements
(3) If LLMs produce test cases, what will be their level[2] (Acceptance Level, System Level, Integration Level or Unit/Code level)?
(4) How to quantify the effectiveness of the produced test cases

ISEC '25, February 20–22, 2025, NIT Kurukshetra, India

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

[1] https://in.mathworks.com/products/simulink-design-verifier.html
[2] The 'Levels' are with respect to V model of the Software Development Life Cycle

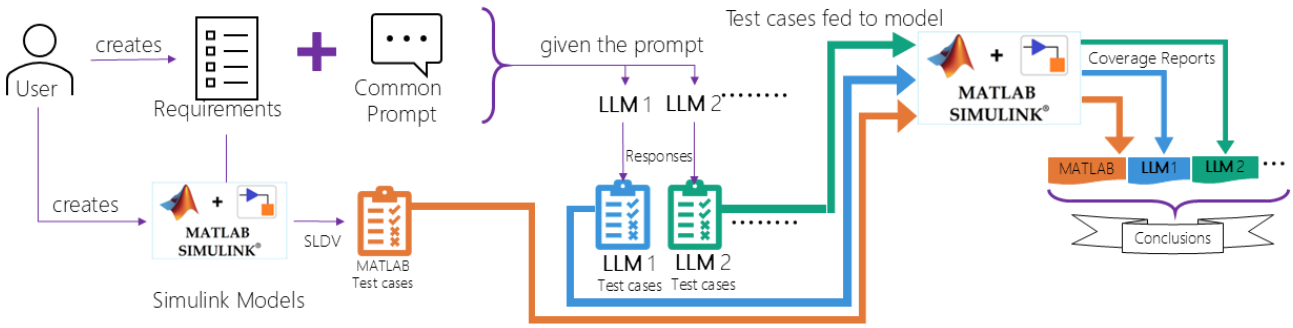

Figure 1: Process used for the study

2 RELATED WORK

Several researchers have argued the need for automated generation of test cases and capabilities of LLMs with respect of test case generation. They have identified several challenges in the test case generation process including validity of generated test cases, deficits in domain-specific knowledge, need for human intervention, overall coverage of the test [5].

LLMs have been used for auto test case generation across a wide variety of domains including distributed control systems (DCS) and programmable logic control systems (PLCs) [10], gaming [15], genetic programming agents [8], medical services [11]. And, among the scenarios where auto test cases generation using LLMs were more successful have been Unit tests [7], conversational agents [3], Graphical User Interface (GUI) [13]. Challenges faced while deploying the LLMs for auto test case generation [5] include paucity of "real-world" datasets, biases and shortcomings in deciphering of intent of a requirement [12], non-availability of formal structured real-world bug reports. It has been observed in prior research that LLMs struggle to generate correct test cases. Researchers have proposed approaches to counter this by fine-tuning the test case generation with further training on specific APIs [15], making the process human-interactive [6], using LLM-based SDKs (software development kits) [16]. However, in all these cases, it could be observed that the input for LLMs is primarily code. In this study, we use textual natural language requirements to get test cases for the final system.

3 EXPERIMENTAL SETUP

For purposes of this study, technical requirements have been specified in natural language. Six general purpose LLMs viz., BARD, ChatGPT3.5, Claude, Gemini, ChatGPT4.o (Omni) and Llama3 were chosen. The key factors for choosing these six LLMs are: public availability, diverse architecture, market relevance, and the variety of potential use cases they support. To minimize bias in the results, a consistent prompt has been utilized across all models, ensuring that the evaluation remains impartial and that the efficacy of each LLM can be fairly assessed. While specialized or contextualized prompts may have produced higher-quality test cases, the primary objective of this study was to facilitate a fair comparison among general-purpose LLMs. Consequently, domain-specific or fine-tuned LLMs were excluded from consideration in this analysis to maintain consistency and impartiality in the evaluation process. This approach allows for a clearer assessment of the capabilities of each selected LLM within the specified parameters.

The responses generated from the prompts were processed and formulated into test cases, which were subsequently utilized to evaluate the software models developed from the corresponding requirements. In parallel, Simulink models were created for the same requirements, and test cases were generated using SLDV. One of the authors possesses industrial experience in working with safety-critical systems and regularly utilizes SLDV as part of their work. Given that the primary objective of the study was to assess the quality of the test cases generated by each of the selected LLMs, it was a logical choice to capture the test results from the six LLMs and compare them with the test cases obtained from SLDV. The test cases generated by SLDV from the MATLAB models serve as the ground truth for this evaluation.

3.1 Experimental Process

The process followed for the comparitive study is shown in Figure 1. This process encompasses the following steps: (1) Requirements Creation, (2) System Model Creation, (3) LLM Test Case Generation, (4) SLDV Test Case Generation, and (5) Coverage Metrics Generation.

3.1.1 Requirements specification. In contemporary systems, various types of requirements exist, each exhibiting different levels of complexity [1]. In this study, the complexity of the requirements is assessed using Cyclomatic Complexity [14] derived from the corresponding Simulink models. Consequently, the authors developed a set of 25 natural language requirements, articulated in English, which vary in complexity. The requirements [3] utilized in this study were manually crafted, drawing from a diverse range of systems regarding scope, construction, domain, and complexity. This approach helps with a wider representation of different contexts and challenges. For example, the requirement number 7 -

"if system wants to activate a test, first check if the test is enabled, if enabled then activate the test, if not enabled then deactivate the test", is a linear requirement and of low complexity, with cyclomatic complexity score of 3.
Where as the requirement number 18 -

[3]Data available at https://doi.org/10.6084/m9.figshare.27020533

Requirement	Complexity	Decision							Condition							MCDC							Execution						
		MATLAB	BARD	ChatGPT3.5	Claude	Gemini	GPT4	llama3	MATLAB	BARD	ChatGPT	Claude	Gemini	GPT4	llama3	MATLAB	BARD	ChatGPT	Claude	Gemini	GPT4	llama3	MATLAB	BARD	ChatGPT	Claude	Gemini	GPT4	llama3
req1	0																						100	100	100	100	100	100	100
req13B	0								100	87.5	100	100	100	100	100	100	0	66.7	100	66.7	100	66.7	100	100	100	100	100	100	100
req19C	0								100	65	65	91.7	100	96.7	90	85.7	0	0	0	57.1	0	64.3	100	100	100	100	100	100	100
req2	0																						100	100	100	100	100	100	100
req3	0								100	100	100	100	100	100	100								100	100	100	100	100	100	100
req6	0								100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
req4	2	100	100	100	100	100	100	100															100	100	100	100	100	100	100
req5	2	100	50	50	50	100	100	50															100	100	100	100	100	100	100
req11	3	100	100	75	100	75	75	75	94.4	66.7	77.8	83.3	69.4	69.4	66.7	83.3	0	16.7	50	8.33	33.3	25	100	100	93.8	100	93.8	93.8	93.8
req13A	3	100	75	100	100	100	100	100	100	100	100	100	100	100	100	100	0	100	100	100	100	100	100	100	100	100	100	100	100
req7	3	100	75	100	100	75	100	100															100	100	100	100	100	100	100
req10	4	100	100	100	100	100	50	50	100	72.2	100	94.4	100	88.9	88.9	100	50	83.3	66.7	100	0	0	100	100	100	100	100	100	100
req12	4	100	83.3	66.7	83.3	83.3	100	100	83.3	75	66.7	75	75	83.3	83.3	66.7	50	16.7	50	50	66.7	66.7	100	100	100	100	100	100	100
req19A	5	100	12.5	100	100	100	12.5	100	100	0	100	100	100	0	100	100	0	60	100	60	0	100	100	50	100	100	100	50	100
req19B	5	100	12.5	87.5	100	100	100	100															100	57.1	100	100	100	100	100
req15	7	100	41.7	66.7	50	66.7	50	50	100	55.6	94.4	77.8	88.9	77.8	77.8	100	0	50	25	0	25	25	100	78.9	89.5	78.9	89.5	78.9	78.9
req16	11	100	100	100	100	100	100	100	100	75	75	75	75	75	75	100	0	0	0	0	0	0	100	100	100	100	100	100	100
req19	11	100	10	90	90	70	100	90	100	90	95	95	95	100	95	100	70	90	80	90	90	80	100	71.9	100	100	93.8	100	100
req20	12	100	100	100	81.8	81.8	72.7	72.7	100	100	100	85.7	85.7	78.6	78.6	100	100	100	100	100	100	100	100	100	100	100	100	100	100
req9	12	100	90	90	90	80	90	90	100	64.3	50	50	42.9	50	50	100	42.9	28.6	28.6	14.3	28.6	28.6							
req8	13	100	95.2	85.7	76.2	81	90.5	85.7																					
req18	17	100	44.4	44.4	44.4	44.4	44.4	44.4	100	77.8	66.7	77.8	77.8	72.2	77.8	100	33.3	22.2	33.3	33.3	33.3	33.3							
req13	25	100	100	100	85.7	92.9	85.7	35.7	100	57.5	57.5	40	55	40	47.5	100	16.7	16.7	5.56	11.1	5.56	0	100	100	100	100	100	100	100
req17	36	100	58.8	64.7	64.7	91.2	47.1	14.7	100	65.2	68.8	73.2	84.8	66.1	28.6	100	21.1	15.8	10.5	50	7.89	0	100	100	100	100	100	100	73.3
req14	61	100	65.9	61	31.7	78	75.6	75.6	100	33.7	32.6	15.1	43	41.9	41.9	100	9.52	7.14	2.38	11.9	11.9	11.9	100	100	100	33.3	100	100	100

Figure 2: Test Coverage Metrics

"The system shall calculate the fuel quantity in the chamber as follows: The mileage of the vehicle is calculated as ..." has 169 words, cyclomatic complexity of 41, and has multiple layers of requirements in which one requirement condition leads to a conditional execution of another condition. The 25 requirements are designed to be independent and not interrelated, ensuring that individual Simulink models can be developed and utilized without dependencies. These requirements focus on low-level system specifications, allowing for the creation of executable system models without necessitating additional models or higher-level systems.

3.1.2 System Model creation. The Simulink system models [3] were manually created for each specified technical requirement, ensuring that each model accurately reflects the associated requirements. Additionally, the model is verified against the requirements manually by co-authors. However, as the number of requirements and complexity of the requirements increases, the feasibility of continuing this manual approach diminishes.

3.1.3 LLM based test case generation. The selected prompt[3] was uniformly applied to all six LLMs in conjunction with each of the specified requirements, and their responses were systematically captured. The exact prompt used for generating test cases was as follows: *Generate exhaustive test cases in table format for the requirement "≪ requirement ≫"*. Any non-generated input is considered as 0 or default (as the case may be). The captured responses are then converted to input format compatible with MATLAB™ toolchain to generate test cases. It is to be noted that there was only one prompt, and it was used across all requirements. A comparison study of different prompts and their result analysis is being planned in another study.

3.1.4 SLDV Test Case Generation. For each requirement, a simulatable and testable Simulink model is developed, ensuring that the model accurately reflects the specified functionalities. Once a Simulink model is constructed, baseline test cases are generated using the SLDV toolbox. SLDV employs formal methods, including static analysis and model checking, to rigorously analyze the Simulink models. This analysis allows SLDV to generate test cases that meet various coverage criteria, such as Decision Coverage, Condition Coverage, and Modified Condition/Decision Coverage. By applying these coverage criteria, SLDV can help effectively evaluate the behavior of the model under different conditions.

3.1.5 Coverage Metrics Generation. All the test cases are provided to the Simulink models and the coverage reports are generated. The focus of this study is only on the coverage of the test cases. The following metrics are considered for this study:

- Decision coverage
- Condition coverage
- MCDC (Modified condition/Decision coverage) [9]
- Execution coverage

The coverage metrics mentioned above are generated by the SLDV toolbox without necessitating the actual code generation for the

Simulink models. However, the study does not focus on other quantitative metrics, such as the correctness, completeness, and precision of the generated test cases. Additionally, aspects related to the efficiency of test case generation, including the time taken to produce these test cases, have also been excluded from consideration. This study aims to assess the effectiveness of test case generation from the perspective of adequate model coverage.

3.2 Evaluation criteria and metrics

The evaluation criteria are as follows: For a given requirement, the LLM-generated test cases are used to get the coverage metrics. These are compared for all requirements. A higher value corresponding to the metric suggests that the LLM is better for that particular coverage category. The cyclomatic complexity of the model generated for the requirement is used as a reference complexity metric. It is to be noted that the cyclomatic complexity is dependent on the type of model-designing methods and practices used in developing the models. It could be treated as subjective, but the authors tried to maintain consistent patterns among all models created to the extent possible.

4 RESULTS AND ANALYSIS

The metric values for the experiment are presented in Figure 2. It is important to note that the MATLAB values are included solely to establish a baseline for the maximum coverage that could potentially be achieved; they should not be directly compared with the results from the LLMs. This distinction arises from the fundamental difference in input types: MATLAB utilizes the model as input, whereas the LLMs generate test cases based on textual requirements. Based on the values achieved the the following observations can be inferred:

LLMs show a decline in performance as the complexity of the requirements increased. In cases where the requirements were linear and of low complexity, all LLMs performed comparably to a commercial testing tool. However, BARD's performance was observed to be the lowest among the six LLMs across the evaluated metrics (figure 3). On an average GPT4.o and Llama3 are also on

		MATLAB	BARD	ChatGPT	Claude	Gemini	GPT4	llama3
Average	Decision	100	69.2	83.2	81.5	85.2	78.6	75.5
	Condition	98.8	71.4	80.5	79.7	82.9	74.4	77.8
	MCDC	96.2	29	45.5	50.1	50.2	41.3	47.1
	Execution	100	93.5	99.2	96	99	96.5	97.5
Median	Decision	100	75	90	90	83.3	90	85.7
	Condition	100	73.6	86.1	84.5	87.3	78.2	81
	MCDC	100	16.7	28.6	50	50	28.6	33.3
	Execution	100	100	100	100	100	100	100

Figure 3: The Median and Average of the LLMs coverage

the lower end among all 4 categories. If requirements are small and not multi-leveled like 13A
"The system shall calculate the final answer C as follows. Var1 is the summation of inputs A and B while Var2 is the summation D and E . C is the summation of Var1 and Var2. When var1 is above 10 then var2 is fixed at 20. When var1 is not above 10 but ver2 is above 20 then var 1 is fixed at 10". The LLMs provide results comparable to SLDV.

There is no clear 'better performer' with increasing complexity in requirements as all LLMs are failing to get more than 25% of MCDC.

Several observations were made regarding the performance of the LLMs in generating test cases for specific requirements where they consistently failed to produce satisfactory results. In instances of multi-level requirements, some LLMs struggle to comprehend the requirement as a cohesive whole ($a- > b- > c- > d == a- > d$) and instead generate test cases at the individual component level. In contrast, some models successfully generate test cases that reflect the hierarchical nature of the requirements (refer req 13A/13B metrics in figure 2).

In particular, when dealing with a single top-level activation control signal, as illustrated in requirement 19A, which states: *"The Engine state is calculated as follows: Only when the key is in ignition the system starts calculating if not the engine will be in off state.When the ignition is on..."*, it was noted that BARD frequently fails to recognize this overarching activation, leading to less effective outcomes compared to ChatGPT (refer to the metrics for requirements 19, 19A, and 19B in Figure 2). Furthermore, for multi-level requirements, it was observed that all LLMs, except for Gemini and Llama 3, tend to overlook the lower-level requirements, as seen in requirement 19C. In the case of requirement 16, which states: *"The system shall do the wing check as follows: The wing 1 will be verified if the wing 1 motor has no failure, and ..."*, all LLMs failed to generate any test cases that achieved coverage for MCDC. This failure can be attributed to the requirement's multi-layered and interdependent nature, wherein each condition is contingent upon the outputs of one or more other requirements. The inherent complexity of the requirements appears to exceed the capacity of the LLMs to effectively analyze and construct valid test cases that would cover the necessary conditions.

The research questions raised in this study could be answered as follows

(1) In regards to LLMs producing test cases using requirements, its definitely possible. However, test case generated did not provide complete coverage. For some of the requirements the test cases did not satisfy the test adequecy criteria.
(2) If the requirements have any complex wording or interdependency or multi layered style , the LLMs fail to generate test cases for those parts
(3) Currently only unit level test cases that correspond to code/model are generated by LLMs. Generic LLMs are not able to generate System and Acceptance level test cases.
(4) The effectiveness of the test cases can be quantified by calculating the overall coverage of the software by the test cases. Note that this does not consider the correctness of the test case itself.

5 LIMITATIONS AND THREATS TO VALIDITY

The requirements in this study are articulated exclusively in English, with word counts ranging from 13 to 429. A total of 25 requirements were constructed with a diverse array of systems in mind, including calculators, cars, airplanes, weather monitoring systems, walking mechanisms, automated teller machines, air conditioning units, and battery charging systems. Each requirement was designed to include at least two inputs and one output.

It is important to note that the correctness of the generated test cases was not evaluated; the primary focus was on coverage metrics. Additionally, only embedded-type requirements were considered for this study. This choice reflects the fact that industrially produced embedded systems typically possess a robust and well-defined set of requirements, particularly in the context of safety-critical systems that must be certified by relevant authorities. In contrast, online and open systems often place less emphasis on stringent requirements, as they are developed within constantly evolving environments where system necessities can change frequently.

Furthermore, advanced requirements involving multiple time bounds, multiple iterations, enumerations, and similar complexities were not included in this evaluation. The intent of the current study is to assess the test generation capabilities of various LLMs, and as this research is in its initial stages, large requirements were not included. This limitation affects the generalizability of the findings.

The complexity metric is based on the cyclomatic complexity score of the developed simulink models. In practice the complexity of a given requirement depends on various technical factors. The Simulink model from the requirement was created and verified for correctness manually. The comparison study conducted here is based on technical requirements created manually and are not part of any actual product. Thus they do not offer a complete picture of what an actual product or system or software requirement might entail. The LLMs may produce different responses for differently worded and structured text even though they possess the same meaning. Additionally, for larger systems, the test cases produced by LLMs often require manual validation by domain experts, adding an additional layer of human involvement, which could diminish the benefit of automating the test case generation process. The prompt was kept constant across all LLMs to assess the LLMs capabilities. Multiple prompts were not attempted to get more contextualized test cases. The motive is to test the LLMs by giving minimum amount of extra information possible. A study can be conducted with better prompt engineering and the results can possibly vary.

6 CONCLUSIONS AND FUTURE WORK

In the present study, we observed some patterns within the responses generated by LLMs, which can be leveraged to formulate requirements that highlight areas of failure. This approach can contribute to a deeper understanding of the models' limitations and facilitate the establishment of a baseline for their performance.

Future research will seek to assess the correctness of the generated test cases, thereby evaluating the capability of each LLM in relation to specific requirements. Expanding the total number of requirements and re-running the experiment with a broader dataset could also yield more comprehensive insights.

Additionally, experimenting with different prompts may uncover further observations and enhance the understanding of how LLMs interpret and generate test cases. To evaluate the practical applicability of test case generation using LLMs, it would be beneficial to apply this methodology to a real-world product with a robust set of requirements.

REFERENCES

[1] Mack Alford. 1980. Software requirements in the 80's. In *Proceedings of the ACM 1980 Annual Conference (ACM '80)*. Association for Computing Machinery, New York, NY, USA, 342–349.

[2] Mohamed Boukhlif, Nassim Kharmoum, and Mohamed Hanine. 2024. LLMs for Intelligent Software Testing: A Comparative Study. In *The 7th International Conference On Networking, Intelligents Systems and Security (NISS 2024)*. ACM, New York, NY, USA, 8.

[3] Pablo C. Cañizares, Daniel Ávila, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2024. Coverage-based Strategies for the Automated Synthesis of Test Scenarios for Conversational Agents. In *5th ACM/IEEE International Conference on Automation of Software Test (AST 2024)*. ACM, New York, NY, USA, 10. https://doi.org/10.1145/3644032.3644456

[4] Hao Ding, Ziwei Fan, Ingo Guehring, Gaurav Gupta, Wooseok Ha, Jun Huan, Linbo Liu, Behrooz Omidvar-Tehrani, Shiqi Wang, and Hao Zhou. 2024. Reasoning and Planning with Large Language Models in Code Development. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24)*. ACM, New York, NY, USA, 11.

[5] Liming Dong, Qinghua Lu, and Liming Zhu. 2024. A Pilot Study in Surveying Data Challenges of Automatic Software Engineering Tasks. In *Proceedings of the 4th International Workshop on Software Engineering and AI for Data Quality in Cyber-Physical Systems/Internet of Things (SEA4DQ '24)*. ACM, New York, NY, USA, 6.

[6] Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madan Musuvathi, and Shuvendu K. Lahiri. 2024. Exploring the Effectiveness of LLM based Test-driven Interactive Code Generation: User Study and Empirical Evaluation. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24)*. ACM, New York, NY, USA, 2.

[7] Vitor H. Guilherme and Auri M. R. Vincenzi. 2023. An initial investigation of ChatGPT unit test generation capability. In *SAST 2023*. ACM, Campo Grande, MS, Brazil. https://doi.org/10.1145/3624032.3624035

[8] Steven Jorgensen, Giorgia Nadizar, Gloria Pietropolli, Luca Manzoni, Eric Medvet, Una-May O'Reilly, and Erik Hemberg. 2024. Large Language Model based Test Case Generation for GP Agents. In *Genetic and Evolutionary Computation Conference (GECCO '24)*. ACM, New York, NY, USA, 10.

[9] Hayhurst Kelly J., Veerhusen Dan S., Chilenski John J., and Rierson Leanna K. 2001. *A Practical Tutorial on Modified Condition/Decision Coverage*. Technical Report. NASA.

[10] H. Koziolek, V. Ashiwal, S. Bandyopadhyay, and K. R. Chandrika. 2024. Automated Control Logic Test Case Generation using Large Language Models. *arXiv preprint arXiv:2405.01874* (2024).

[11] Christoph Laaber, Tao Yue, Shaukat Ali, Thomas Schwitalla, and Jan F. Nygård. 2023. Automated Test Generation for Medical Rules Web Services: A Case Study at the Cancer Registry of Norway. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23)*. ACM, New York, NY, USA, 12.

[12] Huiyuan Lai and Malvina Nissim. 2024. A Survey on Automatic Generation of Figurative Language: From Rule-based Systems to Large Language Models. *ACM Comput. Surv.* 56, 10 (2024), 34.

[13] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2024. Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. ACM, New York, NY, USA, 13.

[14] T.J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2, 4 (1976), 308–320.

[15] Ciprian Paduraru, Alin Stefanescu, and Augustin Jianu. 2024. Unit Test Generation using Large Language Models for Unity Game Development. In *Proceedings of the 1st ACM International Workshop on Foundations of Applied Software Engineering for Games (FaSE4Games '24)*. ACM, New York, NY, USA, 7.

[16] Zafaryab Rasool, Scott Barnett, David Willie, Stefanus Kurniawan, Sherwin Balugo, Srikanth Thudumu, and Mohamed Abdelrazek. 2024. LLMs for Test Input Generation for Semantic Applications. In *Conference on AI Engineering Software Engineering for AI (CAIN 2024)*. ACM, New York, NY, USA, 6.