Unit Test Generation Using Large Language Models: A Systematic Literature Review

Dovydas Marius Zapkus, Asta Slotkienė

Vilnius University, Universiteto g. 3, Vilnius
marius.zapkus@mif.stud.vu.lt, asta.slotkiene@mif.vu.lt

Abstract. Unit testing is a fundamental aspect of software development, ensuring the correctness and robustness of code implementations. Traditionally, unit tests are manually crafted by developers based on their understanding of the code and its requirements. However, this process can be time-consuming, error-prone, and may overlook certain edge cases. In recent years, there has been growing interest in leveraging large language models (LLMs) for automating the generation of unit tests. LLMs, such as GPT (Generative Pre-trained Transformer), CodeT5, StarCoder, LLaMA, have demonstrated remarkable capabilities in natural language understanding and code generation tasks. By using LLMs, researchers aim to develop techniques that automatically generate unit tests from code snippets or specifications, thus optimizing the software testing process. This paper presents a literature review of articles that use LLMs for unit test generation tasks. It also discusses the history of the most commonly used large language models and their parameters, including the first time they have been used for code generation tasks. The result of this study presents the large language models for code and unit test generation tasks and their increasing popularity in code generation domain, indicating a great promise for the future of unit test generation using LLMs.

Keywords: unit test generation, large language model

1 Introduction

Software testing is one of the most important software development processes, which increases the overall quality and reliability of the final product [1].

Unit testing is an essential part of software testing methods. Successful implementation of unit tests can decrease the number of errors in the final product and increase the efficiency of the developer, thus making

DOI: https://doi.org/10.15388/LMITT.2024.20

the software more reliable [2, 9]. Unit tests are used to test units of code, ranging from functions, methods, procedures, etc. [3]

To optimize the testing of information systems, automated testing tools are applied, which help developers save resources and time [2]. Currently, solutions such as Search-Based Software Testing [4] and random test generation [5, 14] are used for software testing. These solutions are capable of generating component tests, but often the generated tests are impractical and difficult to understand [4-6]. Therefore, it has been suggested to use large language models, which would not only effectively cover system functionality but also be clear and easily interpretable. Currently, programs utilizing large language models, such as Github Copilot, can successfully generate code from code comments or complete the remaining part of a started code segment [7, 12]. Consequently, it is believed that large language models can also be employed in unit or component testing.

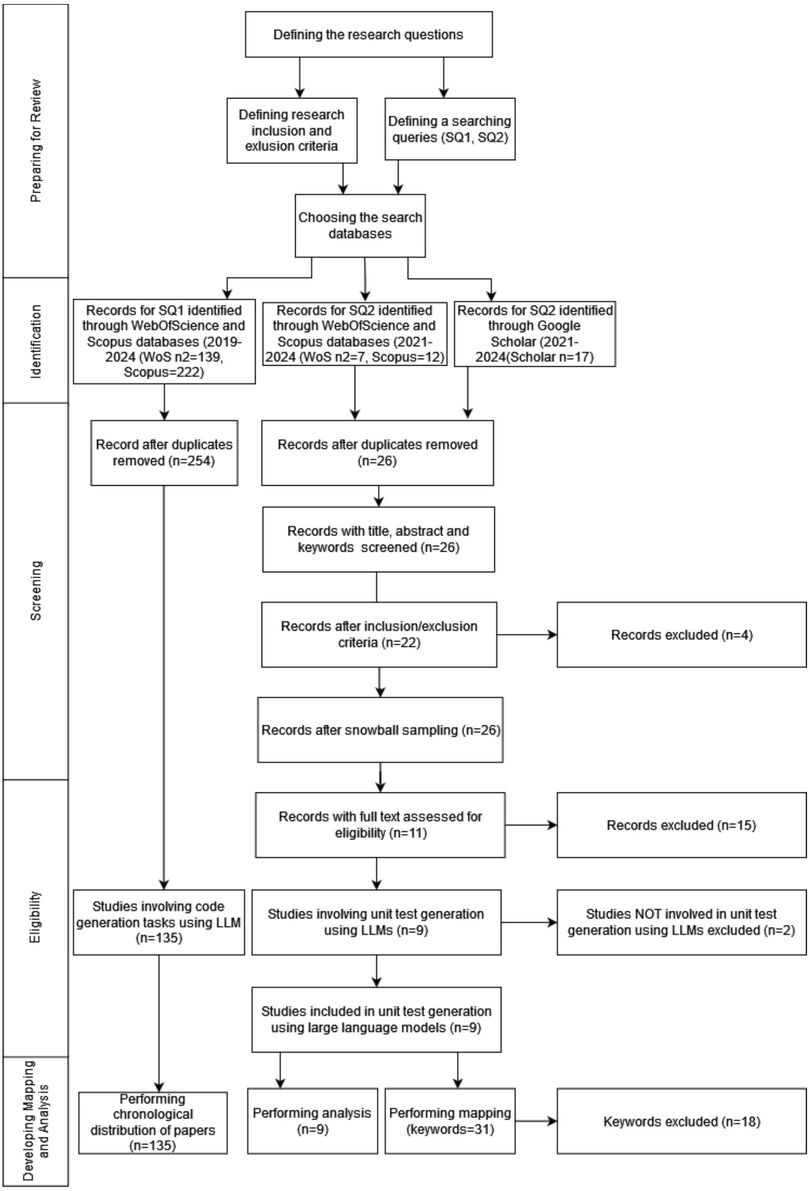
Upon analyzing scientific research [14, 16] on large language models and their generated component tests, it is observed that models such as OpenAI GPT-3 and Codex are the most commonly used LLMs for unit test code generation tasks. These models are already being used in other domains, and their potential and effectiveness in the context of test case generation have not been thoroughly explored.

This paper aims to perform a literature review on studies that analyze LLMs for unit test generation tasks, to identify which large language models are used for these tasks and what are the new relationships between large language models and unit test domains since 2019.

The rest of this paper is structured as follows. Section 2 presents the review methodology. Section 3 discusses the results obtained in the paper. Finally, Section 4 concludes the paper.

2 Research methodology

The review methodology was developed and executed according to the guidelines provided by Kitchenham and Brereton [17, 18]. The structure of the methodology consists of five steps: (1) preparing of review, (2) identification, (3) screening, (4) eligibitation (5) developing mapping and analysis (Figure 1).



Preparing for Review
Identification
Screening
Eligibility
Developing Mapping and Analysis
Defining the research questions
Defining research inclusion and exlusion criteria
Defining a searching queries (SQ1, SQ2)
Choosing the search databases
Records for SQ1 identified through WebOfScience and Scopus databases (2019-2024 (WoS n2=139, Scopus=222)
Records for SQ2 identified through WebOfScience and Scopus databases (2021-2024 (WoS n2=7, Scopus=12)
Records for SQ2 identified through Google Scholar (2021-2024(Scholar n=17)
Record after duplicates removed (n=254)
Records after duplicates removed (n=26)
Records with title, abstract and keywords screened (n=26)
Records after inclusion/exclusion criteria (n=22)
Records excluded (n=4)
Records after snowball sampling (n=26)
Records with full text assessed for eligibility (n=11)
Records excluded (n=15)
Studies involving code generation tasks using LLM (n=135)
Studies involving unit test generation using LLMs (n=9)
Studies NOT involved in unit test generation using LLMs excluded (n=2)
Studies included in unit test generation using large language models (n=9)
Performing chronological distribution of papers (n=135)
Performing analysis (n=9)
Performing mapping (keywords=31)
Keywords excluded (n=18)

Figure 1. The flow diagram of the systematic literature review

In the first step, we raised two research questions, which covered the main research question: *What is covered for unit test generation using large language models?*

RQ1: What is the historical evolution of code (unit test) generation using large language models?

RQ1.1 What kind of large language models are used for code generation?

RQ1.2 Which large language models are used for unit test generation tasks?

RQ2: What are the new relationships between large language models and unit test domains in the analyzed period?

To increase the research accuracy we determined the inclusion and exclusion criteria (Table 1):

Studies Inclusion Criteria (IC)	Studies Exclusion Criteria (EC)
IC1. Studies were published after 2021 **IC2.** The publication must be written in English **IC3.** The publication is a primary study **IC4.** Studies that compare large language models for unit test generation tasks. **IC5.** Studies are in the field of software engineering or computer science **IC6.** Studies which have an empirical background	**EC1**. Studies that are duplicates of other studies of the same authors **EC2**. The reported research does not relate to LLM and unit tests, or the research is not discussed in the context of LLM unit test generation **EC3**. The publication was not written in English **EC4**. Studies not accessible in full-text

Table 1. Inclusion/Exclusion criteria

The search strategy includes two main terms, which help to define the search queries: The final search query is developed based on WoS, Scopus, and Google Scholar databases search requirements as follows:

1. The SQ1 was created and applied as follows: specify the primary search keywords based on the main research question; find substitute alternatives for the large language models (LLMAlternatives) such as GTP-4, Codex, etc.

   SQ1: *LLMAlternatives AND code AND (generation OR automation)*

2. Search query (SQ2) decides RQ1.2 and RQ2 and covers terms related to unit tests: unit test and component test and terms related to large language models: large language model and LLM.

   SQ2: *("unit test*" OR "component test*") AND ("large language model*" OR LLM OR LLM's)*

The identification step was performed the search using the defined search query in three scientific databases such as WoS and Scopus, and Google Scholar (databases set of scientific papers), using these limitations: articles written in only English language (IC2), the document type was only articles, the article should be covered only subject area Computer Science. After searching by selected search query were obtained 37 articles from 2019 to 2023. Because the same search query was performed in three different databases, in the s*creening step, was removed* duplicated studies (11 studies removed) and records were verified by inclusion/exclusion criteria (4 studies excluded). A deeper analysis of the found articles allowed us to notice that some articles we knew about weren't included in the search results. That way, we used snowball sampling and added several scientific articles (4 studies), which included the mentioned terms and the main scientific question. An interim analysis of the keyword map from 9 articles indicated the need to apply a relevancy analysis of full-text articles (eligibility step). The full text of each study is assessed for eligibility (15 excluded) and each study is validated based on whether it performs research on unit test generation using large language models (2 excluded).

3 Results of systematic literature review

In the emerging application of using Large Language Models for generating unit tests and the limited research in this domain, this study instead aims to examine the increasing adoption of LLMs in code generation tasks. By performing a literature review in Section 2 it was observed, that only 9 studies were retrieved. This result wouldn't yield sufficient data to analyze the popularity of LLM's. A more abstract term „code generation" was selected to retrieve a larger number of studies (SQ1), improving the overall accuracy of the chronological distribution of the papers diagram in Figure 2. „Unit test generation" is a subtopic of „code generation tasks" [7], thus we can assume, that LLM's ability to generate code makes it able to generate unit tests. The chronological distribution of papers on LLM for code generation tasks is shown in Figure 2. The papers selected were published from 2019 till the end of 2024 (RQ1).

From Figure 2 we can notice, that the highest number of studies related to LLM and code generation tasks is observed in the year 2023 with a total of 135 studies, contributing to 54.87% of the overall number of studies since the year 2019 (RQ1.1). This indicates that research on LLMs and code generation tasks has been growing in popularity over the years with an average of 56.82% annually. The number of publications for each LLM in Figure 2 indicates, that these models can be used for code generation tasks (RQ1.1). Codex, released in 2021, is the highest-researched LLM for code generation tasks included in 21.9% of all the studies analyzed in Figure 2. Another popular LLM is GPT-2 having an equal number of research papers as Codex, but was released in 2019 and indicates a stagnation or decrease in the number of papers since 2022. Another highly researched LLM is GPT-3, having 50 studies since 2020 and gaining popularity annually. The highest number of LLMs with the ability to generate code was released in 2023.

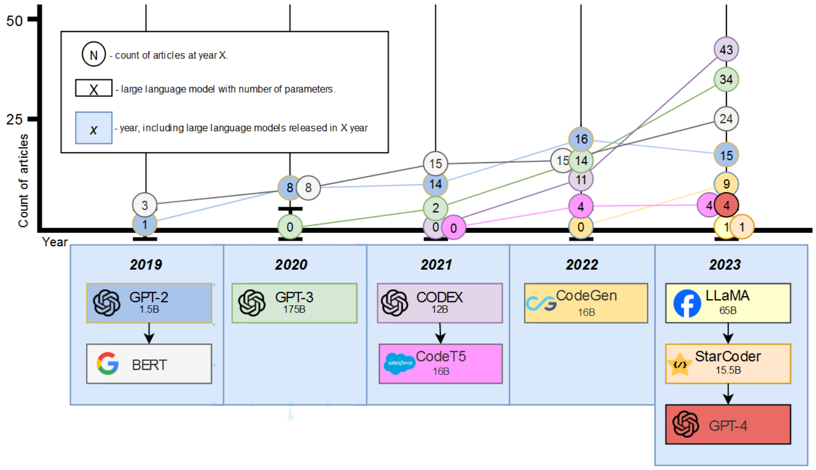


Figure 2. Chronological distribution of papers by large language models used for code generation tasks.

In addition to RQ1.2. From gathered papers [8-16] using the literature review in Figure 1, it was noticed that for unit test generation tasks, models such as Codex, GPT-3, StarCoder, and GPT-4 were used. Figure 2 indicates the rising popularity of LLM‘s ability to generate code. These results indicate that with the rising popularity of LLM, more distinct topics are selected for research of this model, such as unit test generation, while less researched models are excluded from such topics.

For the second research question (RQ2), it was decided to perform a keyword map from articles gathered in Figure 1. The creation of a keyword map was selected for its ability to distinguish associations between different keywords and group these keywords into clusters, thus improving the analysis process. The keyword map was developed by using the VOSviewer tool. The original result contained 49 unique keywords. It was noticed that some of the keywords didn‘t indicate the contextual relationship between LLM and unit tests, these keywords include research type („review“, „literature survey“, „literature review“), abstract keywords („agenda“, „group“, „focus“, „codes“). After removing these keywords, 31 items remained and a keyword map was developed using VOSviewer in Figure 3.

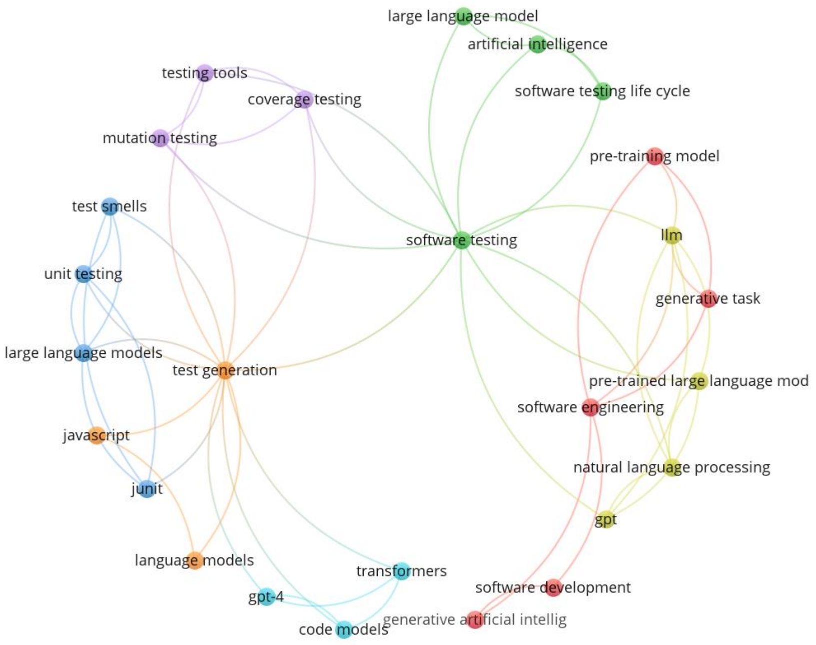


Figure 3. LLM and unit test keywords map

Generated keyword map in Figure 3, aims to answer RQ2. We can see that the 6 clusters were formed. The most frequent keywords were “test generation” and „software testing“. Most associations to other clusters were made through “software testing” keyword. “Test generation” keyword is connected with „gpt-4“, „large language models“, „code models“ keywords, which indicates the type of tools/models that were used for test generation tasks in the studies. “Test generation” keyword also associated with type of unit test quality testing, which is in a color purple cluster. „Software testing“ is associated with “llm” keyword that indicates, that studies were mostly using large language models for testing software.

Summing up, according to the historical overview, the analyzed topic of the unit test generation using large language models is quite new based on the scarcity of articles surrounding it, but it is becoming more and more frequent and since 2023, it has had the highest increase in its relevance. Most studies in recent years have performed research with Codex and GPT-3 large language models.

4 Conclusion and Future Work

This study is a systematic literature review of studies researching large language models and their capability to generate unit tests. It was noticed that LLM and unit test generation topics are new, and related research on these topics is relatively tiny. LLM and code generation tasks have been gaining popularity since 2019, with an average increase of 56.82% annually. From deeper analysis, we can indicate that models such as Codex, GPT-3, StarCoder, and GPT-4 were used for unit test generation tasks. From generating the keywords map, it was noticed that unit test generation has relationships not only with various AI artifacts but also relevant to the quality aspects of the test generation process and test quality.

The results of this study indicate future works in the analyzed area. First, the quality criteria are determined, which helps to evaluate the quality of generated unit tests. The second challenge is research, which shows the activity chain for the test development process using LLMs.

Literature

[1] Nagabushanam, Durga & Dharinya, Sree & Vijayasree, Dasari & Sai Roopa, Nadendla & Arun, Anugu. (2022). A Review on the Process of Automated Software Testing. 10.48550/arXiv.2209.03069.

[2] Job, Minimol. (2021). Automating and Optimizing Software Testing using Artificial Intelligence Techniques. International Journal of Advanced Computer Science and Applications. 12. 10.14569/IJACSA.2021.0120571.

[3] Nikolaeva Zheleva, Dimitrichka. (2021). The role of unit testing in training. In: Development Through Research and Innovation - 2021 [online]: The 2nd International Scientific Conference: Online Conference for Researchers, PhD and Post-Doctoral Students, Au-

gust 27th, 2021, Chişinău: Conference Proceedings. Chişinău, ASEM, 2021, pp. 42-49. ISBN 978-9975-155-54-0.

[4] Fontes, A., & Gay, G. (2023). The integration of machine learning into automated test generation: A systematic mapping study. Software Testing, Verification and Reliability, 33(4), e1845, 10.48550/arXiv.2206.10210.

[5] Hashtroudi, Sepehr & Shin, Jiho & Hemmati, Hadi. (2023). Automated Test Case Generation Using Code Models and Domain Adaptation.

[6] Grano, Giovanni & Palomba, Fabio & Nucci, Dario & Lucia, Andrea & Gall, Harald. (2019). Scented Since the Beginning: On the Diffuseness of Test Smells in Automatically Generated Test Code. Journal of Systems and Software. 156. 10.1016/j.jss.2019.07.016.

[7] Bareiß, Patrick & Souza, Beatriz & d'Amorim, Marcelo & Pradel, Michael. (2022). Code Generation Tools (Almost) for Free? A Study of Few-Shot, Pre-Trained Language Models on Code. 10.48550/arXiv.2206.01335.

[8] Siddiq, M. L., Santos, J., Hasan Tanvir, R., Ulfat, N., Al Rifat, F., & Carvalho Lopes, V. (2023). Exploring the effectiveness of large language models in generating unit tests. arXiv e-prints, arXiv-2305.

[9] Alagarsamy, S., Tantithamthavorn, C., & Aleti, A. (2023). A3test: Assertion-augmented automated test case generation. arXiv preprint arXiv:2302.10352.

[10] Le, H., Wang, Y., Gotmare, A. D., Savarese, S., & Hoi, S. C. H. (2022). Coderl: Mastering code generation through pre-trained models and deep reinforcement learning. Advances in Neural Information Processing Systems, 35, 21314-21328.

[11] Bayrı, V., & Demirel, E. (2023, December). AI-Powered Software Testing: The Impact of Large Language Models on Testing Methodologies. In 2023 4th International Informatics and Software Engineering Conference (IISEC) (pp. 1-4). IEEE.

[12] Huang, Y., Chen, Y., Chen, X., Chen, J., Peng, R., Tang, Z., Huang, J., Xu F. & Zheng, Z. (2024). Generative Software Engineering. arXiv preprint arXiv:2403.02583.

[13] Nguyen-Duc, A., Cabrero-Daniel, B., Przybylek, A., Arora, C., Khanna, D., Herda, T., ... & Abrahamsson, P. (2023). Generative Artificial Intelligence for Software Engineering--A Research Agenda. arXiv preprint arXiv:2310.18648.

[14] Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., & Wang, Q. (2024). Software testing with large language models: Survey, landscape, and vision. IEEE Transactions on Software Engineering.

[15] Guilherme, V., & Vincenzi, A. (2023, September). An initial investigation of ChatGPT unit test generation capability. In Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing (pp. 15-24).

[16] Schäfer, M., Nadi, S., Eghbali, A., & Tip, F. (2023). An empirical evaluation of using large language models for automated unit test generation. IEEE Transactions on Software Engineering.

[17] Kitchenham, Barbara & Brereton, Pearl & Budgen, David & Turner, Mark & Bailey, John & Linkman, Stephen. (2009). Systematic literature reviews in software engineering-A systematic literature review. Information and Software Technology. 51. 7-15. 10.1016/j.infsof.2008.09.009.

[18] Kitchenham, B. and Brereton, P. (2013) A Systematic Review of Systematic Review Process Research in Software Engineering. Information and Software Technology, 55, 2049-2075.