# Using Large Language Models to Generate Concise and Understandable Test Case Summaries

Natanael Djajadi, Amirhossein Deljouyi, Andy Zaidman
*Delft University of Technology*
Delft, The Netherlands
{n.djajadi, a.deljouyi, a.e.zaidman}@tudelft.nl

***Abstract*—Software testing is essential, and automatic test case generation can be an important aid to software engineers. However, generated tests are sometimes difficult to understand. Test summarization approaches that provide an overview of what exactly is tested can provide help, but existing summarization approaches generate documentation that is lengthy and redundant. In this paper, we investigate whether large language models (LLMs) can be used to generate more concise, yet understandable summaries. In a small-scale user study with 11 participants, we obtained positive feedback on the LLM-generated summaries.**

***Index Terms*—Automated Test Generation, Large Language Models, Unit Testing, Readability, Test Case Summarization**

## I. Introduction

Software testing is essential for ensuring software reliability and correctness, but it can be a time-consuming endeavour to write tests [1]–[4]. This has led to initiatives to automate test generation, e.g., EvoSuite [5] and Randoop [6]. While EvoSuite achieves high code coverage, the generated tests often suffer from limited understandability [7]–[10], due to non-intuitive variable naming, meaningful test data, and the absence of informative comments [11], [12].

Several research efforts have aimed at improving test documentation by generating test summaries. One such tool, `TestDescriber` [13], leverages the executed code segments to generate relevant summaries. Another approach, `DeepTC-Enhancer` [14], uses a template-based with combination with deep learning to produce high-level summaries. However, these tools' summaries, as seen in Listings 1, often suffer from issues like redundancy and excessive length.

Recent advancements in large language models (LLMs) have led to investigations into their usefulness for test generation [15]–[19]. In particular, the `UTGen` [20] tool combines `EvoSuite` with LLMs to both explore the search space, thus ensuring high cover coverage, and improve the understandability of the generated test cases through enhanced variable naming, test data, test names, and comments [20].

This paper explores how we can add test summarization to `UTGen` with the use of LLMs. Given the importance of prompt engineering for quality output [21], we investigate how different prompting techniques impact the quality of the LLM-generated outputs. We look into approaches like Chain-of-Thought prompting [22] and few-shot learning [23]. We also compare output quality of a local open-source model, CodeLLama [24], and a commercial closed-source LLM, ChatGPT [25]. We provide a replication package with our implementation, prompts, data, and evaluation results [26].

```
/** A. TestDescriber:
 *  OVERVIEW: The test case "test10" covers around 2.0% (low percentage) of
 *  statements in "ArrayIntList" */

/** B. DeepTC:
 *  1. Creates a new ArrayIntList
 *  2. Adds to "arrayIntList0" 3 times and checks if its size is
 *  3. Expects an IndexOutOfBoundsException when calling removeElementAt
 *  on "arrayIntListo" with argument 1 */
@Test
public void test10() throws Throwable {
    ArrayIntList arrayIntList0 = new ArrayIntList();
    try {
        arrayIntList0.add(0, 0);
        arrayIntList0.add(0, 1);
        arrayIntList0.add(0, 2);
        assertEquals(3, arrayIntList0.size());
        arrayIntList0.removeElementAt((1));
        fail("Expecting exception: IndexOutOfBoundsException");
    } catch (IndexOutOfBoundsException e) {
        // Should be at least 0 and less than 0, found -1
    }
}
```

Listing 1: Examples of test summaries generated by A. TestDescriber and B. DeepTC.

The following research questions steer our investigation:

**RQ1** *What is the impact of using different prompt techniques to generate LLM-based test summaries?*

We investigate how prompt engineering techniques like few-shot prompting and context awareness affect the summaries.

**RQ2** *How do developers rate LLM-generated test summaries compared to those from existing approaches in terms of content, conciseness, and naturalness?*

We perform a user study to compare the quality of LLM-generated summaries with existing tools by looking at the content (completeness and correctness), conciseness (no irrelevant information), and naturalness of the language.

**RQ3** *Which characteristics of LLM-generated test summaries most influence summarization quality?*

We explore how different characteristics shape the quality of LLM-generated summaries, based on developers' feedback.

## II. Background

`TestDescriber` [13] and `DeepTC-Enhancer` [14] both use template-based test case summaries. They fill the template with

the output of SWUM, a technique that uses nouns, verbs, and prepositional phrases to represent program statements [27].

Prompt engineering is important given that each LLM has its favoured input structures. As such, we use action words, adopt a persona, and employ Chain of Thought [18], [22].

Few-shot learning is another technique that can be used to prompt the LLM. Here, one will first give a few examples or demonstrations similar to the task that will be performed, and then prompt the LLM to do the actual task [23].

## III. LLM-generated Summarizations

### A. Prompting Techniques Used for Summarizations

We utilized `codellama:7b-instruct` [28] as the local model, and `GPT-3.5` [29] and `GPT-4o` [30] as commercial models with four prompts utilizing different prompting techniques to generate summaries with the LLM.

*Simple/Baseline:* This basic prompt asks the LLM to generate a summary using only the test method as input. This will be our baseline to see if using different prompting techniques will affect the output of the LLM.

*Prompt Engineering:* Research shows that Chain-of-Thought prompting enhances LLM performance in complex reasoning tasks [22]. As such, we structured our process into five steps, emphasizing generating adequate content, and concise, bullet-point summaries. An example of prompt engineering using the CodeLLama model is shown in Listing 2.

*Context-Awareness:* In addition to the test method used in the simple prompt, we include the Method Under Test (MUT) to give the LLM more context for generating summaries. We hypothesize that this additional context will lead to more detailed explanations of the MUT in the generated summaries.

*Few-shot:* In the few-shot prompt, we provide the LLM with example summaries written by industry developers to illustrate what ideal summaries should look like. After reviewing GitHub repositories, we selected `Spr8510Tests.java` from `Spring` [31], `StanfordCoreNLPServerITest.java` from `CoreNLP` [32], and `LongCycleTest.java` from `Dagger` [33] for their brevity, clarity, and unit test focus. An example of Few-Shot using the GPT-4o model is shown in Listing 2.

### B. Integration in `UTGen`

We integrated the summary functionality in `UTGen`. We also modified the LLM server, where the prompt is processed to generate summaries by connecting to the LLM—either locally for `codellama:7b-instruct` or through the official API for `GPT-3.5` and `GPT-4o`. Additionally, we implemented a prompt healing technique, inspired by token healing [34], to correct formatting deviations and remove unnecessary elements from the generated summaries.

## IV. Experimental Setup

### A. Set-up for user-evaluation (RQ1)

Comparing all LLM summarization techniques with template-based summaries is impractical for a user study. We first select two promising LLM approaches—one open-source and one commercial—based on a preliminary evaluation of various prompt techniques and models.

```java
/** A. GPT-4o - Few-shot
 * Tests adding elements to an ArrayIntList and verifies that removing an
 * element at an invalid index throws an IndexOutOfBoundsException */

/** B. CodeLLama:7b - Prompt Engineering
 * The test case adds 3 elements to the list and asserts that the size is 3.
 * Then, it tries to remove an element at index 1, which is out of bounds,
 * expecting an `IndexOutOfBoundsException`.
 * The test case verifies that the expected exception is thrown with the
 * appropriate message. */
@Test
public void test10() throws Throwable {
    ArrayIntList arrayIntList0 = new ArrayIntList();
    try {
        arrayIntList0.add(0, 0);
        arrayIntList0.add(0, 1);
        arrayIntList0.add(0, 2);
        assertEquals(3, arrayIntList0.size());
        arrayIntList0.removeElementAt((1));
        fail("Expecting exception: IndexOutOfBoundsException");
    } catch (IndexOutOfBoundsException e) {
        // Should be at least 0 and less than 0, found -1
    }
}
```

Listing 2: Examples of LLM-generated test summaries with A. GPT-4o (few-shot) and B. CodeLLama:7b (prompt engineering).

*Experimental Design:* We used two test cases, `ArrayIntList` and `Rational`, from *TestDescriber* [13] and employed three LLMs—`codellama:7b-instruct`, `GPT-3.5`, and `GPT-4o`. Given the non-deterministic nature of these models, we ran each prompt three times. With three LLMs, two tests, four prompt techniques, and three runs, two authors analyzed 72 prompts in total. Our analysis focuses on three criteria—*content* (completeness and accuracy), *conciseness* (absence of irrelevant information), and *naturalness* (flow and tone)—to assess how human-like the generated summaries appeared [13], [14], [35].

*Experimental Procedure:* We used a rule-based evaluation system to reduce subjectivity. Each summary started with a score of 5 on a Likert scale, with deductions for violations. 1) *Content*: Must clearly state what is being tested and how, with deductions for missing or inaccurate details. 2) *Conciseness*: Should be brief, with points deducted for unnecessary information. 3) *Naturalness*: Should have a human-like tone, with deductions for poor flow or formal tone errors.

### B. User Study (RQ2)

We conducted a user study to compare LLM-generated test summaries with those from existing tools. Eleven participants from our network, including eight Bachelor's and three Master's students with at least one year of Java experience (see Table I), completed a survey to assess the quality of these summaries. For the evaluation, we used two test methods from RQ1, along with one test from *DeepTC-Enhancer* and another from *TestDescriber*. This selection enables us to compare the LLM-generated and template-based summaries directly.

*Experimental Procedure:* The survey began with informed consent, followed by four rounds of 20 questions where participants evaluated and compared summaries from two selected LLM approaches, DeepTC-Enhancer, and TestDescriber, each applied to four different test cases. In the first two rounds, participants compared all four tools, while in the last two

| Experience | Industry | Academic |
|---|---|---|
| 0-1 years | 6 (60%) | 3 (30%) |
| 2-3 years | 3 (30%) | 1 (10%) |
| 4-5 years | 2 (20%) | 7 (70%) |
| **Total** | 11 (100%) | 11 (100%) |

TABLE I: Experience of Participants

rounds, they compared the LLM-generated summaries with either DeepTC-Enhancer or TestDescriber.

Participants rated the summarizations based on content, conciseness, and naturalness using a 5-point Likert scale, with optional text boxes for additional feedback. To reduce bias, the order of summaries was randomized in each round, preventing grouping by their respective tools. Additionally, we were present during the evaluation to address questions.

*Analysis:* We apply the Kruskal-Wallis H test ($\alpha = 0.05$) to identify significant differences between groups, as the Shapiro-Wilk test confirmed non-normal data. We use Dunn's test for pairwise comparisons, and Cohen's d for effect size, categorizing the effect as negligible ($d < 0.2$), small ($0.2 \leq d < 0.5$), medium $0.5 \leq d < 0.8$), and large ($d \geq 0.8$) [36].

### C. Elements to explain the results (RQ3)

To answer RQ3, participants described their most and least liked test summary features, such as length and formatting, in two text boxes. Two authors categorized responses using card-sorting method and calculated feature frequencies.

## V. Results

We now discuss the results per research question.

### A. RQ1: Impact of prompt techniques on LLM test summaries

Table II presents the results of a rule-based evaluation of prompt techniques across LLM models, focusing on content, conciseness, and naturalness. The first four columns (Frequency of Points) show the frequency and percentage of scores for each technique, while the last four columns display mean scores for each aspect and an overall average. Overall, *GPT* models consistently outperformed *CodeLlama* across all the techniques. Among the techniques, prompt engineering achieved the best performance, followed by context awareness and few-shot prompting.

For content, simple prompt, context-awareness, and prompt engineering techniques performed similarly, while the few-shot technique scored lower. Few-shot summaries often cover only the main idea without detailing steps, resulting in an average score of around 4 based on our rules. We hypothesize that concise examples lead few-shot prompts to prioritize brevity over completeness. Context-awareness added more detail to the `ArrayIntList` class, while all LLMs consistently included method explanations for the `Rational` class. This may indicate that LLMs may rely on prior training data for these classes or infer methods from test context cues.

In conciseness, *GPT* models outperformed *CodeLlama*. With the few-shot technique, GPT models used one-line examples for concise summaries, while CodeLlama often ignored concise examples, favoring longer explanations. In the prompt-engineering technique, which uses bullet points, CodeLlama frequently repeated lines from test cases, resulting in redundancy. With context-aware prompts, GPT models became less concise, adding details that were sometimes unnecessary. For naturalness, issues like inconsistent tone, repeated numbers (e.g., '3.11041E-4'), and excessive brackets impacted readability. These issues were less evident in prompt-engineering, which follows specific guidelines for summary clarity. The few-shot technique also benefited from examples in GPT models, helping maintain a natural tone.

| Prompt Technique | LLM | Frequency of Points | | | | Mean | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 5 | 4 | 3 | $\leq 2$ | Cont. | Conc. | Nat. | Tot. |
| Simple | CodeL. | 7-19% | 8-22% | 16-44% | 5-14% | 4.1 | 2.9 | 3.4 | 3.5 |
| | GPT3.5 | 24-67% | 11-31% | 1-3% | 0 | 4.8 | 4.6 | 4.5 | 4.6 |
| | GPT4o | 21-58% | 8-22% | 7-19% | 0 | 4.5 | 4.2 | 4.4 | 4.4 |
| Context-Awareness | CodeL. | 17-47% | 6-17% | 11-31% | 2-6% | 4.1 | 3.7 | 4.4 | 4.1 |
| | GPT3.5 | 21-58% | 10-28% | 4-11% | 1-3% | 4.7 | 4 | 4.5 | 4.4 |
| | GPT4o | 19-53% | 13-36% | 4-11% | 0 | 4.7 | 3.9 | 4.7 | 4.4 |
| Prompt Engineering | CodeL. | 14-39% | 13-36% | 7-19% | 2-6% | 4.7 | 3.5 | 4 | **4.1** |
| | GPT3.5 | 23-64% | 13-36% | 0 | 0 | 4.8 | 4.4 | 4.6 | 4.6 |
| | GPT4o | 25-69% | 11-31% | 0 | 0 | 4.8 | 4.7 | 4.6 | 4.7 |
| Few-shot | CodeL. | 5-14% | 14-39% | 4-11% | 13-36% | 3.9 | 2.5 | 3.2 | 3.2 |
| | GPT3.5 | 24-67% | 9-25% | 3-8% | 0 | 3.9 | 4.9 | 4.9 | 4.6 |
| | GPT4o | 25-69% | 11-31% | 0 | 0 | 4.1 | 5 | 5 | **4.7** |

TABLE II: Comparison of different techniques and LLMs

Based on these results, we selected `CodeLlama` with prompt engineering as the open-source model and `GPT-4o` with few-shot as the commercial model for RQ2.

### B. RQ2: Comparative influence of LLM-generated summaries and existing tools

In the first two survey rounds, all summarization approaches—`GPT-4o` (few-shot), `CodeLlama` (prompt engineering), `TestDescriber`, and `DeepTC-Enhancer`—were evaluated together. In the final two rounds, however, LLM-generated summaries were assessed separately due to unique test cases in TestDescriber and DeepTC-Enhancer.

Table III presents results from the first two rounds, where LLM-generated summaries consistently scored above 4 for higher overall ratings and naturalness. Participants preferred these summaries, selecting them 20 and 17 times compared to 7 and 12 for the existing tools. *CodeLlama (prompt engineering)* emerged as the top choice, followed by *GPT (few-shot)*. CodeLlama scored highest in content richness, while GPT was favored for conciseness, indicating participants appreciated both depth and brevity. Figure 1 shows the two final rounds, revealing that: 1) CodeLlama (prompt engineering) received the highest ratings for content quality, 2) GPT-4o (few-shot) scored highest for conciseness, 3) LLM-generated summaries outperformed template-based ones in naturalness.

Two statistical tests comparing LLM-generated summaries with those from existing tools show that:

1) CodeLlama (prompt engineering) and GPT-4o (few-shot) as significantly more concise than TestDescriber (Dunn's test, $p \leq 0.01$, effect sizes 0.92 and 1.41, respectively). GPT-4o (few-shot) also scored significantly higher in naturalness than TestDescriber (p = 0.04, medium effect size 0.60), while no significant difference appeared for content (p = 0.08).

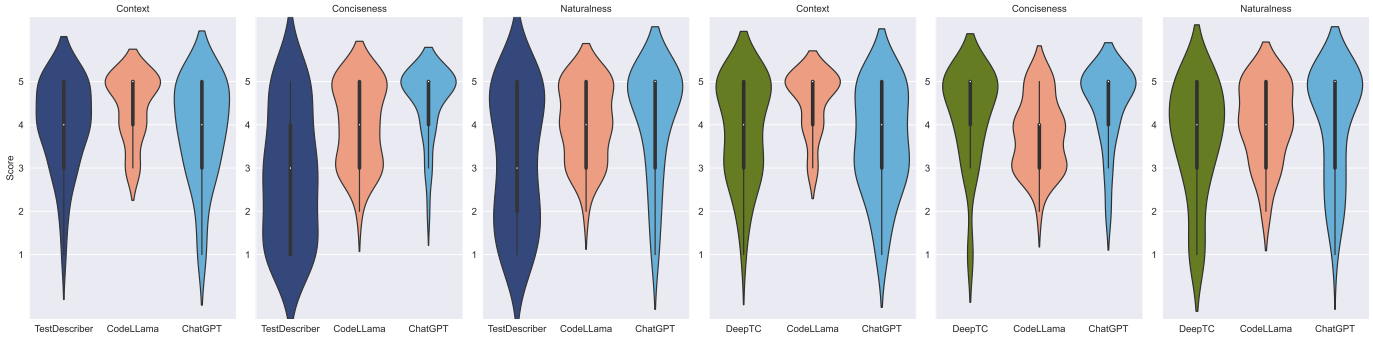


Fig. 1: Results from comparing the summaries of `TestDescriber`, `CodeLlama:7b-instruct`, and `GPT-4o` using a 5-point Likert scale

| Tool | Mean | | | | # times |
|---|---|---|---|---|---|
| | Content | Conciseness | Naturalness | Total | favoured |
| `TestDescriber` | 3.97 | 2.85 | 3.21 | 3.34 | 7 |
| `DeepTC` | 3.88 | 4.21 | 3.64 | 3.90 | 12 |
| `CodeLlama` | 4.48 | 3.93 | 4.09 | 4.17 | 20 |
| `GPT-4o` | 3.78 | 4.57 | 4.05 | 4.13 | 17 |

TABLE III: Left: averages of the quality aspects (content, conciseness, naturalness) of the LLM-generated summaries. Right: comparison of the number of times participants preferred a summary.

2) DeepTC-Enhancer's summaries are rated significantly more concise than CodeLlama (p = 0.03, small effect size 0.46). GPT-4o (few-shot) was rated more concise than CodeLlama ($p \leq 0.01$, medium effect size 0.77), while CodeLlama (prompt engineering) scored higher for content ($p \leq 0.01$, large effect size 0.94).

### C. RQ3: Impact of Characteristics on Test Summary Quality

Analyzing user feedback on both liked and disliked aspects of each approach reveals:

1) *TestDescriber* received positive feedback from two participants for its step-by-step commentary (four mentions) but was criticized by seven participants for excessive detail as reducing readability (twelve mentions). This feedback aligns with its low conciseness scores in Figure 1.

2) *DeepTC-Enhancer* was praised by five participants for concise summaries and numbered steps (seven mentions). However, some participants felt the line-by-line descriptions lacked clarity about the test's purpose, leading to lower content scores than *CodeLlama (prompt engineering)*.

3) *CodeLlama (prompt engineering)* was highlighted by eight participants for its detailed explanations (fourteen mentions) and skimmable format (five mentions). However, five participants noted redundancy (seven mentions), which affected conciseness relative to DeepTC-Enhancer and GPT-4o.

4) *GPT (few-shot)* was commended by five participants for concise summaries (six mentions) and by another five for clear test objectives (eight mentions). However, seven participants found it lacked detail (seventeen times), which affected content scores, as seen in Figure 1.

## VI. Discussion

### A. Revisiting the Research Questions

Our findings in exploring the research questions reveal that the prompt engineering technique produces richer content, while few-shot prompting provides more concise summaries. Context-aware prompting also benefits complex tests by understanding test logic. Among models, GPT models consistently outperformed CodeLlama:7b (*RQ1*). Comparing LLMs with tools like TestDescriber and DeepTC-Enhancer highlighted variability in quality metrics like content and conciseness (*RQ2*). For example, *GPT-4o (few-shot)*'s summaries were concise but sometimes lacked detail, affecting clarity. Characteristics like inline comments and summary length varied in their influence, with conciseness sometimes beneficial and other times limiting (*RQ3*). These mixed results suggest that no single approach or characteristic element is universally optimal for test case summarization. We also expect the performance gap between GPT and CodeLlama to widen with further evaluation on more test cases.

### B. Threats to Validity

*Construct Validity* could be affected by our self-assessment of summaries in RQ1, introducing subjective bias despite our consistent guidelines.

*Internal Validity* was addressed by randomizing summaries and anonymizing tool identifiers to reduce bias. Permutation testing indicated no significant effect of participant background on ratings, supporting our findings' robustness.

*External Validity* is limited by our selection of test cases: we used two classes for RQ1 and four tests for RQ2 and RQ3 to avoid skewing results. A broader user evaluation would also improve generalizability.

## VII. Conclusions and Future Work

Our research reveals that while LLM-generated test summaries often excel in quality, particularly in creating concise summaries, their effectiveness depends on the right combination of prompting techniques and LLM choice. We evaluated four prompting strategies across three LLMs, finding that *codellama:7b-instruct* excelled with prompt engineering, and *GPT-4o* outperformed other ChatGPT models in few-shot prompting by balancing content, conciseness, and naturalness. In user evaluations, LLM-generated summaries outperformed existing tools (*TestDescriber* and *DeepTC-Enhancer*), particularly in naturalness, though summary length remains a factor affecting readability. For future work, we intend to perform a user study to deeply explore the differences in prompt techniques across LLMs.

## REFERENCES


[1] M. Aniche, C. Treude, and A. Zaidman, "How developers engineer test cases: An observational study," *IEEE Trans. Software Eng.*, vol. 48, no. 12, pp. 4925–4946, 2022.

[2] M. Beller, G. Gousios, and A. Zaidman, "How (much) do developers test?" in *37th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2015, pp. 559–562.

[3] M. Beller, G. Gousios, A. Panichella, and A. Zaidman, "When, how, and why developers (do not) test in their IDEs," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2015, pp. 179–190.

[4] M. Beller, G. Gousios, A. Panichella, S. Proksch *et al.*, "Developer testing in the IDE: patterns, beliefs, and behavior," *IEEE Trans. Software Eng.*, vol. 45, no. 3, pp. 261–284, 2019.

[5] G. Fraser and A. Andrea, "EvoSuite: Automatic test suite generation for object-oriented software," Proceedings of the Joint Meeting on Foundations of Software Engineerin (ESEC/FSE), pp. 416–419.

[6] C. Pacheco and M. D. Ernst, "Randoop: Feedback-directed random testing for java," in *Conf. on Object-Oriented Programming Systems and Applications (OOPSLA-Companion)*. ACM, 2007, pp. 815–816.

[7] A. Arcuri, "An experience report on applying software testing academic results in industry: we need usable automated test generation," *Empir. Softw. Eng.*, vol. 23, no. 4, pp. 1959–1981, 2018.

[8] C. E. Brandt, A. Khatami, M. Wessel, and A. Zaidman, "Shaken, not stirred: How developers like their amplified tests," *IEEE Trans. Software Eng.*, vol. 50, no. 5, pp. 1264–1280, 2024.

[9] C. E. Brandt and A. Zaidman, "Developer-centric test amplification," *Empir. Softw. Eng.*, vol. 27, no. 4, p. 96, 2022. [Online]. Available: https://doi.org/10.1007/s10664-021-10094-2

[10] C. E. Brandt, M. Castelluccio, C. Holler, J. Kratzer *et al.*, "Mind the gap: What working with developers on fuzz tests taught us about coverage gaps," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. ACM, 2024, pp. 157–167.

[11] B. Li, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and N. A. Kraft, "Automatically Documenting Unit Test Cases." International Conference on Software Testing, Verification and Validation (ICST): IEEE, 2016, pp. 341–352.

[12] A. Deljouyi and A. Zaidman, "Generating understandable unit tests through end-to-end test scenario carving," in *Proceedings of the 23rd IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2023, pp. 107–118.

[13] S. Panichella, A. Panichella, M. Beller, A. Zaidman, and H. C. Gall, "The impact of test case summaries on bug fixing performance: an empirical investigation," Proceedings of the 38th International Conference on Software Engineering (ICSE), 2016, pp. 547–558.

[14] D. Roy, Z. Zhang, M. Ma, V. Arnaoudova *et al.*, "DeepTC-enhancer: improving the readability of automatically generated tests," Proceedings of the International Conference on Automated Software Engineering, 2021, pp. 287–298.

[15] M. Schäfer, 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, 2024.

[16] S. Yu, C. Fang, Y. Ling, C. Wu, and Z. Chen, "LLM for test script generation and migration: Challenges, capabilities, and opportunities," *arXiv*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2309.13574

[17] J. Wang, Y. Huang, C. Chen, Z. Liu *et al.*, "Software testing with large language model: Survey, landscape, and vision," *arXiv*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.07221

[18] A. Deljouyi, "Understandable test generation through capture/replay and llms," in *Proceedings of the International Conference on Software Engineering (ICSE-Companion)*. ACM, 2024, pp. 261–263.

[19] K. El Haji, C. E. Brandt, and A. Zaidman, "Using GitHub Copilot for test generation in Python: An empirical study," in *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST)*. ACM, 2024, pp. 45–55.

[20] A. Deljouyi, R. Koohestani, M. Izadi, and A. Zaidman, "Leveraging large language models for enhancing the understandability of generated unit tests," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2025.

[21] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, "LLM is like a box of chocolates: the non-determinism of ChatGPT in code generation," *arXiv*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.02828

[22] J. Wei, X. Wang, D. Schuurmans, M. Bosma *et al.*, "Chain of thought prompting elicits reasoning in large language models," 01 2022. [Online]. Available: https://www.researchgate.net/publication/358232899_Chain_of_Thought_Prompting_Elicits_Reasoning_in_Large_Language_Models

[23] N. Nashid, M. Sintaha, and A. Mesbah, "Retrieval-based prompt selection for code-related few-shot learning," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2450–2462.

[24] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla *et al.*, "Code llama: Open foundation models for code," *arXiv*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.12950

[25] "Introducing ChatGPT," https://openai.com/index/chatgpt/, accessed: 2024-06-10.

[26] LLM-TestSummaries, "Replication package of llm-generated summaries," Nov. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.14133029

[27] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the International Conference on Program Comprehension (ICPC)*. ACM, 2014, pp. 279–290.

[28] "Codellama-7b-instruct," https://ollama.com/library/codellama:7b-instruct, accessed: 2024-06-01.

[29] T. B. Brown, B. Mann, N. Ryder, M. Subbiah *et al.*, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[30] OpenAI:, J. Achiam, S. Adler, S. Agarwal *et al.*, "Gpt-4 technical report," *arXiv*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774

[31] S. Projects, "Spr8510tests.java," https://github.com/spring-projects/spring-framework/blob/35474915903970c410ea61f737ee2aeeab41e063/spring-web/src/test/java/org/springframework/web/context/support/Spr8510Tests.java#L78, 2024, accessed: 2024-06-01.

[32] S. NLP, "Stanfordcorenlpserveritest.java," https://github.com/stanfordnlp/CoreNLP/blob/246007929ca8461804d2241b5b3ccbc9897eb1bd/itest/src/edu/stanford/nlp/pipeline/StanfordCoreNLPServerITest.java#L164, 2024, accessed: 2024-06-01.

[33] Google, "Longcycletest.java," https://github.com/google/dagger/blob/9a67471586c3ed1bccb5d0e13a86af06e024cd1a/javatests/dagger/functional/cycle/LongCycleTest.java#L37, 2024, accessed: 2024-06-01.

[34] G. Dagan, G. Synnaeve, and B. Rozière, "Getting the most out of your tokenizer for pre-training and domain adaptation," https://arxiv.org/abs/2402.01035, 2024.

[35] L. Moreno, J. Aponte, G. Sridhara, A. Marcus *et al.*, "Automatic generation of natural language summaries for java classes," in *Proceedings of the International Conference on Program Comprehension (ICPC)*. IEEE, 2013, pp. 23–32.

[36] G. Sullivan and R. Feinn, "Using effect size—or why the p value is not enough," *Journal of graduate medical education*, vol. 4, pp. 279–82, 09 2012.