Readability Driven Test Selection

Using Large Language Models to Assign Readability Scores and Rank Auto-Generated Unit Tests

Ismaël Zaidi

Supervisors: Prof.dr. A. (Andy) E. Zaidman, A. (Amir) Deljouyi

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Ismaël Zaidi
Final project course: CSE3000 Research Project
Thesis committee: Prof.dr. A. (Andy) E. Zaidman, A. (Amir) Deljouyi, Dr. A. (Asterios) Katsifodimos

Abstract

Writing tests enhances quality, yet developers often deprioritize writing tests. Existing tools for automatic test generation face challenges in test understandability. This is primarily due to the fact that these tools fail to consider the context, leading to the generation of identifiers, test names, and identifier data that are not contextually appropriate for the code they are testing. Current metrics for judging the understandability of unit tests are limited as they do not take into account contextual factors such as the quality of comments. Developing a metric to evaluate test readability is essential for selecting the most comprehensible tests. This research builds on UTGen, incorporating LLMs to enhance the readability of automatically generated unit tests. We developed a readability score and used LLMs to evaluate and rank tests, comparing these rankings with human evaluations. This research concludes that LLMs can successfully evaluate the readability of test cases. The GPT-4 Turbo Simple Prompt model exhibited the best performance, with a correlation of 0.7632 with human evaluations. Through comparing different LLMs and techniques for assigning readability scores, we identified approaches that closely matched human evaluations, demonstrating that LLMs can successfully rate the readability of test cases.

1 Introduction

According to CISQ the cost of poor software quality in the United States in 2022 has been determined to be at least 2.41 trillion dollars [2]. This shows the magnitude of the problem of poor software quality and the importance of having good software quality. Writing tests has shown to improve software quality [3]. Even with this knowledge developers perceive writing tests as mundane and prioritize other tasks [10].

Tools have been developed to automatically generate tests, demonstrating satisfactory coverage [9]. However, these tools still face significant challenges, particularly regarding the understandability and readability of the generated tests. For example, they generate identifiers, test names and identifier data that do not make sense based on the context of the code they are testing, making it difficult for developers to comprehend the purpose of the tests and verify their functionality. Studies such as the one by Grano et al. [7] highlight these issues. Additionally, writing tests is a dynamic process. Once written, these tests will be used by other developers to understand the code and assist in debugging. Therefore, they must be clear and easy to understand.

For instance, consider a test case in Listing 1 where the identifier names, data, and test naming is random and lack meaningful context. While the test might be functionally and structurally correct, its readability suffers, making it more difficult for developers to understand and maintain.

Having an automated readability ranking is crucial because it ensures that the selected generated tests are not only functionally correct but also easy to understand and maintain. The automated ranking also saves time by eliminating the need for extensive manual review, allowing developers to focus on other tasks. To achieve automated ranking, we first need a robust and accurate method for scoring the readability of tests.

```java
@Test
@Timeout(value = 4000, unit = TimeUnit.MILLISECONDS)
public void tstXY12A45() throws Throwable {
    JSWeaponData abc123 = new JSWeaponData();
    SimpleWeapon def456 = new SimpleWeapon(-2704, "3]B;R`L#}d:!", 696, 696);
    boolean ghi789 = abc123.isGoldenGun(def456);
    assertFalse(ghi789);
}
```

Listing 1: Motivating Example

1.1 Background and Related work

Readability metrics used by studies such as [5] and [4] tend to yield high ratings since they primarily evaluate structural aspects like the presence of assertions, line length, and the number of identifiers. These metrics do not account for contextual elements such as meaningful test naming and identifier naming, which are crucial for understanding the test's purpose and functionality. This means that current readability metrics would give Listing 1 a high readability score, regardless of the context. However, giving Listing 1 a high readability score is not correct since it clearly lacks in readability. LLMs can address these limitations by analyzing and understanding the context of the code. LLMs can evaluate code aspects, such as the quality of test names and identifiers. In addition, it can capture other contextual elements of readability. This capability enables LLMs to effectively score test cases, ensuring that the tests are not only structurally sound but also contextually comprehensible.

A. Deljouyi also recognized these issues and developed UTGen [1], a tool that integrates search-based software testing with LLMs to improve the understandability of automatically generated test cases. The purpose of this paper is to build on UTGen by addressing the challenge of enhancing the readability of automatically generated tests. We hypothesize that by developing a readability score for unit tests and using LLMs to evaluate and rank these tests, we will achieve scores that align closely with human evaluations conducted by ourselves and other researchers.

1.2 Research Questions

This research is steered by the following research question and sub-questions:

RQ: How can LLMs be utilized to assign readability scores and rank automatically generated unit tests based on their readability?

SQ$_1$: *How can existing readability metrics from software engineering and natural language processing be adapted to develop an LLM-based algorithm for evaluating and scoring the readability of unit test code?*

In current research on the readability of software tests, features such as test and identifier names, comments, test summaries, length, and assertions have been recognized as

[1]paper has not yet been published

important to understanding [5], [11] and [4]. These features and general aspects are used in combination with a LLM to generate readability scores.

SQ$_2$: *How accurately can LLMs assess the readability of unit tests compared to human evaluations?*

Human evaluation is essential to establish a reliable benchmark for readability. Since readability is inherently subjective and context-dependent, human judgment provides the nuanced understanding necessary to evaluate test cases accurately. Comparing LLM assessments with human evaluations ensures that the LLMs' scores are aligned with human perception, validating the effectiveness of the LLM-based approach. Since humans will eventually be the ones using the tool, their understandability is crucial to ensure that we can eventually select the most readable tests for them.

1.3 Contributions

The main contributions of this research lie in the development and validation of an approach for ranking and evaluating the readability of automatically generated unit tests using LLMs. By adapting existing readability metrics from both software engineering and natural language processing domains, we construct an LLM-based algorithm capable of evaluating and scoring the readability of unit test code. By leveraging the contextual understanding capabilities of LLMs, we enhance these metrics to account for meaningful test naming and identifier naming, making the evaluations even more robust. Through an evaluation comparing LLM assessments with human evaluations, we aim to ascertain the accuracy of LLMs in assessing test readability.

2 Implementation

The aim of this approach is to determine if LLMs can be used to evaluate and rank unit tests effectively. We will be using Java as the primary programming language for this implementation. The implementation process involves several key steps. Initially, test suites are generated, and a script is used to extract all the unit tests from them. These extracted tests are then fed into an script, which sends requests to the LLM using a predefined prompt. The LLM responds with readability scores for each test, which are processed by an analysis script. These processed scores are then used to conduct further analyses, allowing us to evaluate the effectiveness of the LLM in ranking the unit tests based on readability.

2.1 Feature Selection

Readability is inherently a subjective matter, often influenced by individual preferences and experiences. This subjectivity poses a challenge when translating readability criteria into prompts for LLMs. Recognizing this complexity, we reviewed existing research to identify the factors that most consistently impact readability, both positively and negatively.

Our investigation revealed several key factors that contribute to readability. Initially, we focused on more specific elements such as identifiers, test names, and comments. These factors were chosen based on their direct influence on particular aspects of code readability according to [5], [11] and [4]:

- **Identifiers**: The conciseness and descriptiveness of variable and function names play a crucial role in how easily a piece of code can be understood.
- **Test Names**: Well-chosen test names provide immediate context and intent, aiding in the comprehension of the test's purpose.
- **Comments**: Clear and informative comments help explain the purpose and functionality of code sections, making the code more understandable.

These elements served as our initial metrics, providing a focused approach to evaluating specific parts of the code.

However, we recognized the need to expand our scope to include more general factors that contribute to an overall sense of readability. Consequently, we incorporated additional metrics that reflect broader, more holistic aspects of code quality:

- **Conciseness**: The brevity of code without sacrificing clarity. Concise code is typically easier to read and maintain.
- **Completeness**: This metric assesses whether the code fully addresses the requirements it is intended to meet, ensuring that tests are thorough and comprehensive.
- **Naturalness**: Refers to the code's resemblance to human language, making it more intuitive and easier to follow. Natural code flow enhances readability by reducing cognitive load [1].

By categorizing these metrics into a specific and a general part, we developed a dual-faceted approach to readability assessment. The specific metrics allow for a granular evaluation of particular aspects of the code, while the general metrics provide a broader overview of the code's overall readability. This methodology enables us to leverage LLMs more effectively, translating the nuanced concept of readability into actionable prompts and evaluations.

2.2 Prompt Generation

For the prompting structure, we took inspiration from A. Deljouyi [2], adapting and refining their approach to suit our specific needs. Initially, we used a baseline prompt to rank a test without specific guidelines, establishing a control measure for readability scoring. This baseline can be seen in Listing 2.

[2]paper has not yet been published

```
[INST] As a Java developer, your task is to evaluate the readability of the
↪  provided Java code.
Please follow these steps:
1. Carefully read the Java code provided between the [CODE] tags.
2. Assess the readability of the code on a scale from 1 to 5, where 1 means
↪  very difficult to understand and 5 means very easy to understand.
3. Indicate your readability score by placing it between the [SCORE] and
↪  [/SCORE] tags.
Your goal is to provide an evaluation solely focused on the readability of
↪  the code.[/INST]
Respond only with: The score of the code is: [SCORE]X[/SCORE]
[CODE]{method_body}[/CODE]
```

Listing 2: Baseline Prompt

Improving the baseline implementation, we refined our approach by focusing on two key aspects of a test: specific features and general features. The specific features include identifiers, test names, and comments, while the general features encompass conciseness, completeness, and naturalness. This dual focus allowed us to separately assess both specific and general readability factors, thus enhancing the overall quality of the readability scores.

To guide the evaluation process, we provided the LLM with detailed instructions on what the scores (from 1 to 5) represent for each factor and asked it to score these factors accordingly. See Listings 3 and 4 for illustrative examples. The new structure includes an introduction which resembles the first lines of Listing 2 and the feature descriptions with corresponding scores from 1 to 5. Clear instructions on what to respond were used, making it easier to extract the scores. We use the [SCORE]x[SCORE] brackets for this purpose, as demonstrated in Listing 2.

Additionally, we utilized a few-shot learning technique according to the techniques described by N. Nashid [8] to further refine the prompt, providing the LLM with examples of both well-written and poorly written test cases to improve its understanding of readability criteria. This was done for both the specific aspects and the general aspects. An example of a bad example and a good example can be seen in Listing 5 and Listing 6.

```
b. Test Name: Assess the descriptiveness and clarity of test method name.
  1: Test name is vague or does not reflect the purpose of the test (e.g.,
↪    test1).
  2: Test name is somewhat unclear or generic.
  3: Test name is fairly descriptive but could be more specific.
  4: Test name is clear and mostly descriptive.
  5: Test name is highly descriptive and clearly reflects the purpose of
↪    the tests.
```

Listing 3: Specific Example

```
c. Naturalness: Evaluate the ease with which the code reads, similar to
↪  natural language, making it easy to understand.
  1: Code is difficult to read and understand, with poor structure and
↪    readability.
  2: Code is somewhat difficult to read, with unclear or convoluted
↪    expressions.
  3: Code is reasonably readable but could be more natural and fluid.
  4: Code is mostly readable and easy to follow.
  5: Code reads naturally and is very easy to understand, resembling
↪    natural language.
```

Listing 4: General Example

```
#### Bad Example ####

[CODE]
public void t2() {{
    // Testing registration
    u usr = new u("jd", "pw", "jd@ex.com");
    boolean regRes = as.reg(usr);
    assertTrue(regRes);

    u retUsr = us.fu("jd");
    assertNotNull(retUsr);
    assertEquals("jd", retUsr.gU());
}}
[/CODE]

- Conciseness: 2. The code is somewhat concise but has unclear
↪  abbreviations making it less readable.
- Completeness: 3. The code is fairly complete but could cover more aspects
↪  or edge cases.
- Naturalness: 2. The code is somewhat difficult to read, with unclear or
↪  convoluted expressions.

[SCORE]2.33[/SCORE]
```

Listing 5: Few Shot - Unreadable Example

```
#### Good Example ####

[CODE]
public void testUserLogin() {{
    // Test logging in with valid credentials
    User user = new User("john_doe", "password123");
    boolean loginResult = authService.login(user);
    assertTrue(loginResult);

    // Test logging in with invalid credentials
    User invalidUser = new User("invalid_user", "wrong_password");
    boolean invalidLoginResult = authService.login(invalidUser);
    assertFalse(invalidLoginResult);
}}
[/CODE]

- Identifier: 5. Identifiers like 'user', 'loginResult', 'authService', and
↪  'invalidUser' are concise, highly descriptive, and enhance readability.
- Test Name: 5. The test name 'testUserLogin' is highly descriptive and
↪  clearly reflects the purpose of the test.
- Comments: 5. Comments are comprehensive, clear, and enhance understanding
↪  of the code.

[SCORE]5.00[/SCORE]
```

Listing 6: Few Shot - Readable Example

2.3 Execution

In this section, we detail the process of utilizing the created prompts with multiple LLMs for our readability assessment. We employ various LLMs for this implementation, including CodeLlama 7B parameter version, ChatGPT 3.5, and ChatGPT 4. The execution process involves several key steps to ensure accurate and efficient readability evaluation.

The process works as follows:

1. **Retrieving Test Cases**: Initially, a script retrieves all the test cases that require evaluation. This script scans the generated test suites and extracts individual test cases, preparing them for readability assessment.
2. **Sending Prompts to Models**: Another script is responsible for sending the appropriate prompts to the selected LLM model. The script utilizes the predefined prompts based on our identified readability factors, ensuring each test case is evaluated according to our criteria.
3. **Extracting Readability Scores**: After the LLM processes the prompts, a script extracts the readability scores from the model's response. These scores reflect

the LLM's evaluation of the test cases based on the provided readability metrics.

The scripts are designed with flexibility in mind, allowing us to easily choose both the LLM model and the specific prompt used for evaluation. This modular approach enables us to switch between different LLMs and prompts, facilitating testing and comparison of readability assessments across various models. By automating these steps, we ensure a consistent and scalable method for evaluating the readability of unit tests. This systematic execution process leverages the power of LLMs to provide detailed readability scores, enhancing our ability to assess and improve the quality of automatically generated test cases.

3 Experimental Setup

The experimental setup for this study is designed to provide a replicable methodology, detailing the specific conditions under which the experiments were conducted. The goal of this evaluation is to understand how humans score the readability of code, specifically examining the scores they give to the specific and general aspects of the code. By doing this, we aim to compare these human evaluations with the LLM scores to gauge their alignment and accuracy.

3.1 User Evaluations

The datasets used in this experiment consisted of various code snippets sourced from established human-written repositories and generated tests from EvoSuite and UTGen. Experiments were conducted in an online environment. The experimental procedure began with questions about the background of the participants, namely the years of Java testing experiences and highest/current level of education. Participants were sourced from students at TU Delft, all studying Computer Science and Engineering at the bachelor's or master's level. The participant distribution was as follows: 8 bachelor's students, 3 master's students, 4 participants with 0-1 years of Java testing experience, 1 participant with 1-3 years of experience, and 6 participants with 3-5 years of experience. None of the participants had more than 5 years of Java testing experience. After the background questions, participants were presented with the tests including the methods under test and asked to rank the readability of each snippet on a continuous scale from 1 to 5. See Listing 7 for an example.

The code snippets used in the study included 10 tests:

1. Three tests from EvoSuite, each with varying difficulties (easy, medium, and hard) in terms of readability.
2. Three tests from UTGen, each with varying difficulties (easy, medium, and hard) in terms of readability.

```
@Test
public void testPLZAddressCombination() {
    // Given
    Customer customer = new Customer("204", "John Do", "221B Bakerstreet");
    when(addressService.getPLZForCustomer(customer)).thenReturn(47891);
    // When
    String address = customerService.getPLZAddressCombination(customer);
    // Then
    assertThat(address, is("47891_221B Bakerstreet"));
}
```

Listing 7: Survey Test Example

3. Two tests sourced from CustomerServiceTest.java (Test 1 and Test 2).
4. Two tests sourced from StringUtilsTestUnit5.java (Test 3 and Test 4).

This resulted in a total of 10 code snippets. Participants evaluated each snippet one by one, also having access to the methods under test. For each snippet, the first question asked participants to give a readability score on a continuous scale, which served as a baseline scoring. After this initial assessment, participants were asked to rank specific aspects of the snippet: identifier quality, test naming quality, and comment quality, on a scale from 1 to 5. They then evaluated general aspects: conciseness, completeness, and naturalness, also on a scale from 1 to 5. After all the snippets, participants were asked to rank the importance of these aspects from most important to least important.

The duration of the experiment was 15-20 minutes per participant. Participants joined an online call where the procedure was explained, emphasizing that they should rank based on their first impression intuitively, resembling a real-world scenario where generated tests are reviewed quickly. Data collection was facilitated using the Qualtrics platform.

3.2 Data Collection and Analysis

Data collection involved human evaluations, with the primary metrics for evaluation including readability scores and factor-specific ratings. Tools for data collection included the online survey platform Qualtrics for human evaluations. The experimental data was analyzed using statistical methods, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and correlation to compare readability scores from human evaluations and LLM responses. Correlation analysis was conducted to explore the relationships between human scores and LLM scores.

3.3 Replicability

To ensure the replicability of our experiments, detailed instructions are provided, including access to scripts for test extraction and LLM processing, and guidelines for conducting human evaluations.

The steps to replicate the experiment are as follows:

1. **Create Survey with 10 Tests**: Prepare a survey containing the 10 code snippets, which include three tests from EvoSuite, three tests from UTGen (each with easy, medium, and hard readability), and four tests from public repositories.
2. **Include 3 Types of Questions + Final Question**: Design the survey to include questions about specific features, general features, and a final question ranking the importance of these aspects.
3. **Source Participants from CS Students**: Recruit participants who are students studying Computer Science and Engineering at the bachelor's or master's level.
4. **Ensure 15-20 Minutes Duration**: Make sure the survey takes approximately 15-20 minutes to complete.

5. **Explain Ranking Intuitively**: Conduct an online call to explain to participants that they should rank the readability based on their first impression, mimicking a real-world scenario where tests are reviewed quickly.

4 Results

The results of this study are organized to provide a clear and detailed presentation of our findings. The results are structured according to the sub-questions and hypotheses outlined at the beginning of our study. Data are presented using tables, figures, and graphs.

SQ1: LLM-based algorithm for scoring readability

In Section 2, we discussed the prompt generation and feature choices. Having implemented these, we needed a way to see if the readability metrics work. The evaluation works in two ways: first, by comparing our scores to a different study, and second, by comparing them to our own user evaluation. The latter will be discussed under subquestion 2.

For the first comparison, we refer to the study conducted by A. Deljouyi. In their study, they compared the readability scores for EvoSuite and UTGen and found from user evaluations that UTGen was more readable. To verify the soundness of our method, we used the same test set with the GPT-3.5 simple prompt and analyzed the results.

Figure 1 shows a boxchart of the UTGen and EvoSuite readability scores. The results indicate that UTGen test cases were rated significantly higher in readability compared to EvoSuite by both the LLMs and human evaluators. This supports the effectiveness of our LLM-based readability scoring method, aligning with Deljouyi's findings and validating our approach.

SQ2: LLM evaluations vs human evaluations

The user evaluation results, presented in Table 1, provide an overview of how different test types and names performed in terms of readability. The evaluations are divided into two main score types: specific and general. For each test, an average score (Avg) was calculated based on user evaluations, reflecting the perceived readability of the generated tests. This average score includes the rankings for the specific aspects (identifier quality, test naming quality, and comment quality) and general aspects (conciseness, completeness, and naturalness), with the scores for each aspect first added and then divided by 3. Additionally, an overall average score (Overall) was provided for each test type, derived from the first question of every snippet, which served as the overall readability baseline.

To further substantiate the effectiveness of our readability model, we compared the scores given by the LLMs with the user evaluations. Figures 2 and 3 show box plots of the specific and general scores, respectively, for the different models. The consistency between the LLM evaluations and human evaluations provides confidence in the reliability of the LLM-based readability scoring.

Table 1: User Evaluation Results

TestType	TestName	ScoreType	Avg	Overall
EvoSuite	Easy	specific	2.97	4.02
EvoSuite	Easy	general	3.97	4.02
EvoSuite	Medium	specific	2.72	2.85
EvoSuite	Medium	general	3.33	2.85
EvoSuite	Hard	specific	3.00	3.45
EvoSuite	Hard	general	3.52	3.45
UTGEN	Easy	specific	4.61	4.83
UTGEN	Easy	general	4.73	4.83
UTGEN	Medium	specific	4.37	4.42
UTGEN	Medium	general	4.24	4.42
UTGEN	Hard	specific	4.27	4.45
UTGEN	Hard	general	4.36	4.45
OtherTests	Test1	specific	2.39	2.08
OtherTests	Test1	general	2.85	2.08
OtherTests	Test2	specific	3.70	3.95
OtherTests	Test2	general	4.06	3.95
OtherTests	Test3	specific	3.73	3.98
OtherTests	Test3	general	3.82	3.98
OtherTests	Test4	specific	3.37	3.57
OtherTests	Test4	general	3.32	3.57

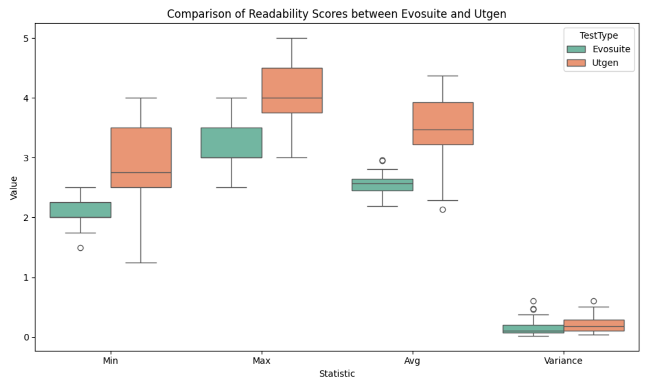

Figure 1: Comparison of Readability Scores: EvoSuite vs. UTGen

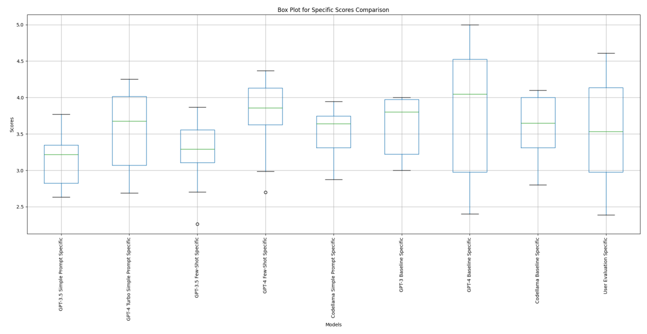

Figure 2: Box-chart Specific Aspects Models

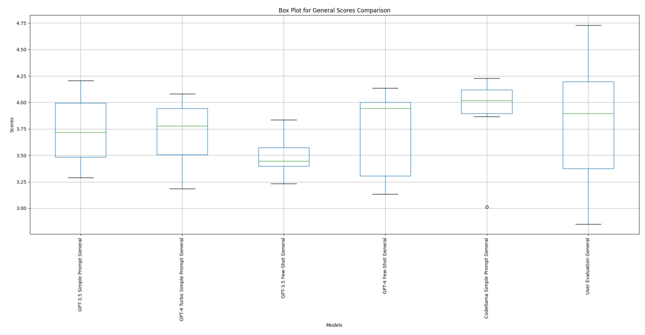

Figure 3: Box-chart General Aspects Models

Table 2 presents the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Correlation between the LLM evaluations and user evaluations for specific scores. The GPT-4 Turbo Simple Prompt model exhibited the best performance with an MAE of 0.3754, an RMSE of 0.4607, and a correlation of 0.7632, indicating a strong agreement with human evaluations.

Model	MAE	RMSE	Corr
GPT-3.5 Simple Prompt	0.5063	0.6351	0.7122
GPT-4 Turbo Simple Prompt	0.3754	0.4607	0.7632
GPT-3.5 Few-Shot	0.6386	0.7457	0.3629
GPT-4 Few-Shot	0.4678	0.5841	0.6566
Codellama Simple Prompt	0.5508	0.6764	0.3336

Table 2: MAE, RMSE, and Correlation for Specific Scores

Table 3 presents the same metrics for general scores. The GPT-3.5 Simple Prompt model performed best with an MAE of 0.2886, an RMSE of 0.3583, and a correlation of 0.8021.

Model	MAE	RMSE	Corr
GPT-3.5 Simple Prompt	0.2886	0.3583	0.8021
GPT-4 Turbo Simple Prompt	0.4207	0.5503	0.2668
GPT-3.5 Few-Shot	0.4659	0.5750	0.5755
GPT-4 Few-Shot	0.4294	0.5655	0.3028
Codellama Simple Prompt	0.4712	0.5575	0.2858

Table 3: MAE, RMSE, and Correlation for General Scores

Finally, Table 4 compares the baseline models. The Codellama Baseline, despite being a weaker model overall, performed the best among the baseline models with an MAE of 0.4420, an RMSE of 0.5705, and a correlation of 0.6089.

Model	MAE	RMSE	Corr
GPT-3 Baseline	0.5870	0.6935	0.3442
GPT-4 Baseline	0.7310	0.8577	0.5099
Codellama Baseline	0.4420	0.5705	0.6089

Table 4: MAE, RMSE, and Correlation for Scores Comparison with User Evaluation for Baseline Models

The high correlation values between LLM evaluations and human evaluations, particularly for the simpler prompt models, underscore the reliability of our LLM-based readability scoring. The results demonstrate that LLMs can effectively replicate human judgments in assessing the readability of test cases, providing a robust method for automating this aspect of software testing.

5 Discussion

The discussion section interprets our results, offering insights into the implications of our findings, and suggesting areas for further research. This section is critical for articulating the value of our research. Reflecting on the main findings, our results address the research questions and hypotheses stated in the introduction, providing valuable insights into the readability of code snippets generated by different methods and LLMs. We observed that LLMs can effectively and correctly evaluate tests, which is consistent with our initial hypothesis.

SQ1: LLM based algorithm for scoring readability

For the first comparison, we refer to the study conducted by A. Deljouyi. In their study, they compared the readability scores for EvoSuite and UTGen and found from user evaluations that UTGen was more readable. To verify the soundness of our method, we used the same test set with the GPT-3.5 simple prompt and analyzed the results. Our readability metric also indicated that UTGen is indeed more readable, thus confirming the findings of the Deljouyi study and validating our approach. As shown in Figure 1, the improved UTGen version scores higher on readability for the same tests compared to EvoSuite. This consistency reinforces the robustness of our LLM-based readability scoring.

SQ2: LLM evaluations vs human evaluations

To further substantiate the effectiveness of our readability model, we analyzed the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Correlation between the LLM evaluations and user evaluations for specific and general scores. These metrics were chosen for their ability to quantify the accuracy (MAE), the magnitude of error (RMSE), and the strength of relationship (Correlation) between LLM-generated scores and human evaluations.

The box plots in Figures 2 and 3 reveal that user evaluations have more variance than most of the LLM models. This indicates the inherent subjectivity of readability as perceived by different users. Despite this variance, the LLM models show consistent performance.

In terms of specific features of the test cases, the simple prompts generally exhibited the most correlation with human evaluations, with GPT-4 Turbo Simple Prompt performing the best. This is evidenced by the lowest MAE (0.3754), RMSE (0.4607), and highest correlation (0.7632) for specific scores, indicating a strong agreement between the model's readability assessments and human judgments.

For general aspects, the GPT-3.5 Simple Prompt had the lowest MAE (0.2886) and RMSE (0.3583), and the highest correlation (0.8021). This suggests that, for general readability features, the simpler prompts might be more effective in mirroring human evaluations.

Interestingly, the Codellama Baseline performed the best among baseline models despite being considered a weaker model overall. It achieved an MAE of 0.4420, an RMSE of 0.5705, and a correlation of 0.6089, which indicates that it has a considerable ability to align with human readability assessments.

The consistency between LLM evaluations and human evaluations provides confidence in the reliability of our LLM-based readability scoring. The high correlation values indicate a strong agreement between the LLMs and human evaluators, suggesting that the LLMs can be reliably used to assess readability. Any observed differences in specific cases can be attributed to factors such as the training data used for the LLMs or the evaluation criteria employed by human users. This leads us to the research question:

RQ: How can LLMs be utilized to assign readability scores and rank automatically generated unit tests based on their readability?

Our findings demonstrate that LLMs can be effectively utilized to assign readability scores and rank automatically generated unit tests by incorporating both structural and contextual elements into the scoring algorithm. By adapting readability metrics to include specific features such as identifier quality, test naming quality, and comment quality, as well as general features like conciseness, completeness, and naturalness, LLMs can provide a comprehensive evaluation of test readability. The high correlation between LLM scores and human evaluations indicates that LLMs are capable of accurately reflecting human judgment, thus enabling the automated ranking of unit tests based on readability. This automated ranking not only ensures that the tests are functionally correct and easy to understand but also significantly reduces the need for extensive manual review, allowing developers to focus on other critical tasks.

Implications

The findings from this research demonstrate that readability metrics, when combined with LLMs, can closely approximate human evaluations of test readability. By leveraging LLMs for readability assessment, we can prioritize the selection of tests that are not only functional but also easily comprehensible.

Our research highlights the effectiveness of LLMs in conjunction with readability metrics. Notably, the simple prompts used with the GPT models showed the most promise, indicating their potential for correctly scoring the readability of automatically generated tests.

6 Conclusion

This study demonstrated that LLMs, particularly when using simple prompts with OpenAI's GPT models, can effectively evaluate and rank the readability of automatically generated unit tests. The strong alignment between LLM scores and human judgments highlights the potential of LLMs in this domain. These findings show promise for future research to further refine and expand the applications of LLMs in improving test readability.

Key Findings

1. Validation of Readability Metrics:

- Our readability metrics, developed using LLMs, were validated by comparing their assessments with a previous study by A. Deljouyi. The consistency in results confirms the reliability of our metrics.
- Specifically, the comparison showed that the improved UTGen test suites were more readable than those generated by EvoSuite, as indicated by both LLM evaluations and human judgments.

2. Performance of LLM Models:

- Among the models tested, GPT-4 Turbo Simple Prompt exhibited the best performance for specific readability features with the lowest Mean Absolute Error (MAE) of 0.3754, Root Mean Squared Error (RMSE) of 0.4607, and highest correlation of 0.7632.
- For general readability features, GPT-3.5 Simple Prompt achieved the lowest MAE of 0.2886, RMSE of 0.3583, and highest correlation of 0.8021, suggesting that simpler prompts may be more effective in these contexts.
- Interestingly, the Codellama Baseline, despite being considered a way weaker model overall, performed best among the baseline models with an MAE of 0.4420, RMSE of 0.5705, and a correlation of 0.6089. This highlights the potential of even less advanced models to provide valuable readability assessments.

3. Subjectivity of Readability:

- Analysis of the box plots indicated that user evaluations exhibited more variance compared to the LLM models, underscoring the subjectivity inherent in readability assessments.
- Despite this variance, LLM models showed consistent performance, validating their use for this purpose.

Future Work

Several areas could be further investigated:

- **Expansion to Other Types of Tests:** Future studies could apply our readability evaluation approach to other types of tests and programming languages to validate its effectiveness across different contexts.
- **Broader Dataset and LLM Configurations:** To enhance the generalizability of our findings, future research should consider a broader range of datasets and LLM configurations.
- **Integration into Automated Test Generation Systems:** Future work should explore how readability rankings evaluated by LLMs can be effectively integrated into automated test generation systems to enhance the readability of unit tests.
- **Adjusting Metric Weights:** Another avenue for future research is to explore the use of different weights for the readability metrics. By assigning different levels of importance to each metric, researchers can tailor the readability assessments better.

In conclusion, this study highlights a promising direction for leveraging LLMs to score the readability of automatically generated test cases.

7 Responsible Research

Throughout this research, we have adhered to the standards of ethical conduct and ensured the reproducibility of our methods in line with the Code of Conduct for Research Integrity. This includes a commitment to not manipulating, fabricating, or selectively trimming data. All steps in our research process have been documented to sustain scientific integrity.

To enhance transparency and reproducibility, we have maintained a spreasheet of all experiments conducted. This details every aspect of the experimental setup, procedures followed, and data collected. All datasets used in our study are stored securely and, where possible, made accessible to the broader research community. By making our data accessible, we aim to facilitate the verification and extension of our work by other researchers.

Moreover, we have embraced the FAIR (Findable, Accessible, Interoperable, and Reusable) framework [6] in managing our research data. By adhering to these principles, we ensure that our data is not only accessible but also well-documented and structured in a way that other researchers can easily find, use, and build upon.

Overall, our commitment to ethical research practices and the reproducibility of our methods underscores the robustness and reliability of our findings. We believe that by adhering to these principles, we contribute to the advancement of knowledge in a responsible and transparent manner.

References

[1] Casey Casalnuovo, Kevin Lee, Hulin Wang, Prem Devanbu, and Emily Morgan. Do programmers prefer predictable expressions in code? *Cognitive science*, 44 12:e12921, 2020.

[2] Consortium for Information & Software Quality (CISQ). The cost of poor quality software in the us: A 2022 report, 2022. Accessed: 2023-04-22.

[3] L. Crispin. Driving software quality: How test-driven development impacts software quality. *IEEE Software*, 23, 2006.

[4] Ermira Daka. *Improving readability in automatic unit test generation.* PhD thesis, 07 2018.

[5] Ermira Daka, José Campos, G. Fraser, Jonathan Dorn, and Westley Weimer. Modeling readability to improve unit tests. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[6] GO FAIR. Fair principles, 2024. Available: https://www.go-fair.org/fair-principles/ [Accessed: 12-Jun-2024].

[7] Giovanni Grano, Simone Scalabrino, H. Gall, and R. Oliveto. An empirical investigation on the readability of manual and generated test cases. *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 348–3483, 2018.

[8] Noor Nashid, Mifta Sintaha, and Ali Mesbah. Retrieval-based prompt selection for code-related few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2450–2462, 2023.

[9] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering*, 44:122–158, 2018.

[10] Philipp Straubinger and Gordon Fraser. A survey on what developers think about testing. *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 80–90, 2023.

[11] D. Winkler, P. Urbanke, and R. Ramler. What do we know about readability of test code? - a systematic mapping study. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1167–1174, Honolulu, HI, USA, 2022.