Assessing Evaluation Metrics for Neural Test Oracle Generation

Jiho Shin, Hadi Hemmati, Moshi Wei, Song Wang

Abstract—Recently, deep learning models have shown promising results in test oracle generation. Neural Oracle Generation (NOG) models are commonly evaluated using static (automatic) metrics which are mainly based on textual similarity of the output, e.g. BLEU, ROUGE-L, METEOR, and Accuracy. However, these textual similarity metrics may not reflect the testing effectiveness of the generated oracle within a test suite, which is often measured by dynamic (execution-based) test adequacy metrics such as code coverage and mutation score. In this work, we revisit existing oracle generation studies plus *gpt-3.5* to empirically investigate the current standing of their performance in textual similarity and test adequacy metrics. Specifically, we train and run four state-of-the-art test oracle generation models on seven textual similarity and two test adequacy metrics for our analysis. We apply two different correlation analyses between these two different sets of metrics. Surprisingly, we found no significant correlation between the textual similarity metrics and test adequacy metrics. For instance, *gpt-3.5* on the *jackrabbit-oak* project had the highest performance on all seven textual similarity metrics among the studied NOGs. However, it had the lowest test adequacy metrics compared to all the studied NOGs. We further conducted a qualitative analysis to explore the reasons behind our observations. We found that oracles with high textual similarity metrics but low test adequacy metrics tend to have complex or multiple chained method invocations within the oracle's parameters, making them hard for the model to generate completely, affecting the test adequacy metrics. On the other hand, oracles with low textual similarity metrics but high test adequacy metrics tend to have to call different assertion types or a different method that functions similarly to the ones in the ground truth. Overall, this work complements prior studies on test oracle generation with an extensive performance evaluation on textual similarity and test adequacy metrics and provides guidelines for better assessment of deep learning applications in software test generation in the future.

Index Terms—AI4SE, Large Language Model, Neural Oracle Generation, Test Adequacy Metrics, Automated Testing, and Test Case Generation

1 Introduction

Unit testing is considered a standard procedure when developing a software product. The primary goal of unit testing is to verify whether a unit behaves as expected. However, writing an effective unit test requires non-trivial effort for developers. To mitigate this challenge, there have been numerous recent studies to automate the process exploiting deep neural generation models [1]–[4]. These models do not consider the specific state (behavior) they want to test or verify (oracle) and mainly focus on catching higher-level exceptions or crashes of the software. Recently, researchers further propose test oracle generators based on deep learning and large language models (LLMs), which aim to generate meaningful assertions or exception-handling logic for a method under test by providing the state they want to verify [5]–[9]. We call these approaches neural oracle generation (NOG).

Textual similarity metrics such as BLEU [10] and exact match accuracy [11] have been widely adopted to evaluate NOG models' performance. These metrics are calculated automatically and statically based on the textual similarity of the generated oracle and the ground truth oracle. Due to the ease of collecting benchmarks (no test case execution is needed) and the cost efficiency in the calculation (string similarity calculation), most studies adopt these metrics to evaluate the performance of generated test oracles. Although these generic metrics are useful to evaluate the quality in readability and maintainability of tests (similarity with human-written), their benefit in special domains such as test adequacy of unit test cases and test oracle generation is unclear.

Although the current field of NOG only considers textual similarity metrics, test adequacy metrics, i.e. code coverage and mutation scores, were widely used to evaluate generated test cases and oracles [12], [13]. These metrics evaluate the effectiveness of unit tests and oracles in their completeness of test coverage and fault detectability. Test adequacy metrics are dynamic, meaning they are only obtainable by executing the software under test and the test suite thus cannot be evaluated using static/automated textual similarity metrics. These are the currently well-known metrics that can evaluate the actual capability of tests in the literature [14]–[16].

Problem Statement: Despite the wide use of textual similarity metrics in evaluating NOGs, no study confirms their correlation with the generated oracle's testing capabilities. Without such a study, there is no guarantee that a generated oracle with higher BLEU (but not 100%) is

- *J. Shin is with Lassonde School of Engineering, York University, Canada. E-mail: jihoshin@yorku.ca*
- *H. Hemmati is with Lassonde School of Engineering, York University, Canada. E-mail: hemmati@yorku.ca*
- *M. Wei is with Lassonde School of Engineering, York University, Canada. E-mail: moshiwei@yorku.ca*
- *S. Wang is with Lassonde School of Engineering, York University, Canada. E-mail: wangsong@yorku.ca*

Manuscript received October 19, 2023; revised -, 2023.

"better" (i.e., covering the same functionality and potentially detecting the same bug as the ground truth oracle) than one with lower BLEU. The oracle might be textually similar to ground truth, but functionality-wise, it verifies a completely different assertion. In other words, if the textual similarity (i.e., BLEU type metrics) and the test adequacy metrics have a low correlation, it would be a problem to only consider the textual similarity metric as the primary evaluation criterion for NOGs. This paper aims to fill this gap and comprehensively analyze the relationship between dynamic test adequacy metrics and static textual similarity metrics for NOGs.

```
// Test prefix
testFileManagerNoFile () {
   FileManager fileManager = FileManager.create();
   fileManager.addLocatorFile();
   try {
      InputStream in =
           fileManager.open(filenameNonExistent);
      closeInputStream(in);
      "<AssertPlaceHolder>";
   } catch(NotFoundException ex) {
   }
}
// Focal method
private void closeInputStream(InputStream in ) {
   try {
      if ( in != null ) in.close ( ) ;
   } catch ( Exception ex ) {
   }
}

// Ground truth oracle
assertNull("Found non-existent file: " +
     filenameNonExistent, in)

// Generated oracle by GPT-3.5
assertEquals(null, in)
```

Listing 1: An example of oracle generation.

Listing 1 shows an example of when these two metrics do not agree with each other. The listing shows an input and output example of an oracle generated by *gpt-3.5*, i.e., an LLM developed by *OpenAI* which is a widely used generative model in various domains including software engineering [4], [17]. The task of a NOG is to generate an appropriate oracle that will be substituted with "`<AssertPlaceHolder>`" when an aggregation of the test prefix and the focal method is given as input. As we can observe, the generated oracle and the ground truth oracle function have the same functionality (i.e., check whether '`in`' is null) but differ in the type of assert statement used. We can consider that the model generated a correct oracle with the same functionality and testing capability leading to the same code coverage and mutation score. However, the textual similarity metrics cannot capture their capability resulting in a low BLEU score of 0.21. Such examples call for a more thorough investigation of using textual similarity metrics and their correlation to test adequacy metrics in evaluating oracle generation tasks.

Before delving into the correlation between the two metrics, we first investigate current NOG literature to find their performance in various textual similarity metrics, i.e. BLEU, CodeBLEU, ROUGE-L, METEOR, Accuracy, Edit Similarity, and ChrF. Then we check the testing effectiveness of the oracles by measuring the test adequacy metrics, i.e., line coverage and mutation score. After, we calculate the correlations between the pairs of evaluation metrics (seven textual similarity metrics and two dynamic test adequacy metrics). Finally, we perform an ablation study on some of the results to further analyze why we observe such discrepancies in the two types of metrics.

The results show that the correlation between these two evaluation metric categories is insignificant. Our manual analysis found cases where the static metrics show an oracle is textually similar to a developer-written oracle but are quite different semantically (their line coverage and mutation scores are very different). Or vice versa, the static metrics suggest the two oracles are syntactically very different, but semantically very similar (e.g., same line coverage and mutation score). These disagreements were the root cause of the low correlation between the two metrics. Thus, we claim that researchers should not only consider textual similarity metrics assuming a high correlation between the textual similarity metrics and the test adequacy metrics.

In summary, our main contributions are as follows:

- We revisit the performance of four state-of-the-art neural oracle generation models on two categories of metrics, i.e., static textual similarity metrics and dynamic test adequacy metrics.
- As far as we know, this paper is the first to show that there is no significant correlation between the two categories of metrics, which shapes future research in this domain.
- We provide a manual analysis of the findings and provide justifications on why there is a mismatch between static and dynamic metrics in this domain, based on our observations.
- We provide a new benchmark for evaluating and studying NOG models, with almost 30K executable methods under test and their corresponding metrics from 13 open-source projects. We also release our experiments' dataset and source code to help other researchers replicate and extend our study[1].

The rest of this paper is organized as follows. Section 2 presents the background. Section 3 shows the experimental setup. Section 4 presents the evaluation results. Section 5 discusses the threats to the validity of our study. Section 6 presents the related studies. Section 7 concludes this paper.

2 Background

2.1 Current NOG's Evaluations

The current literature on NOG mainly focuses on the JUnit frameworks, a `Java` unit testing framework. Here we list the most closely relevant studies in NOG that generate assertion oracles when the test prefix and method under test are given. *ATLAS* [5] was the first study to exploit a deep neural generative model for generating assertion oracles. They have evaluated their models in two different settings: 1) the abstract data where they normalize all identifiers, e.g. IDENT_0/1/2 and METHOD_0/1/2, and 2) the raw data where they keep all the raw identifiers. They have used exact match accuracy and BLEU-4 score to evaluate the performance. They also investigated what types of assertions were generated by the models, e.g. `asserEquals`, `assertNull`,

1. https://github.com/shinjh0849/assessing_ntog

TABLE 1: Performance of NOGs in the original papers.

NOGs	BLEU	ACC
ATLAS	61.85	17.66
IR Old	**78.86**	46.54
IR New	60.92	42.20
TOGA	-	**69.00**

`assertThat`, etc. Yu and Lou et al. [6] extended the former by integrating simple information retrieval techniques (abbr. *IR*), i.e. Jaccard coefficient [18], Overlap [19], and Dice coefficient [20]. They have used the same evaluation metrics from the former, i.e. accuracy and BLEU-4. *TOGA* [7] exploits a unified transformer-based neural model to generate exception handling and assertion for unit test case oracles. It first generates oracle candidates using type constraints for assertion types and forms vocabularies by observing the variables accessible from the test for the parameters. Then it ranks the best candidate using CodeBERT [21], choosing the top-1 as the generated oracle. These proposed methods have shown great potential in NOGs using textual similarity metrics, ranging from 17.66% [5] to as high as 96% *TOGA* in accuracy according to their reports. In Table 1, we report the static textual similarity metrics used in the relevant studies, i.e., BLEU and accuracy score.

2.2 Test Adequacy Metrics

In software testing [12], [13], the effectiveness of a unit test or an oracle is evaluated using test adequacy metrics, e.g., code coverage and mutation score. Code coverage metrics evaluate how much the software under test is exercised [22], [23] by the test case. When software test suites have low coverage, some parts of the software remain untested, leading to incomplete testing. However, high coverage (e.g., covering all lines of code) does not guarantee that the test cases detect all bugs in that code. Different code coverage metrics exist, e.g., function, line, statement, condition, branch, path, data-flow coverage, etc. For example, if a test case exercises 5 lines of software under test with a total number of 10 lines, the line coverage of the test case will be 50%.

Another category of test adequacy metrics is fault-based coverage criteria. Unlike coverage-based metrics, fault-based metrics focus on detecting faults (defects). A test is effective if it finds all defects in the code. This is the ultimate metric for any testing approach, but given that the defects are typically not known while testing, in practice fault-based testing must simulate defects. There are many approaches for this for example seeding old bugs into code or manually adding likely bugs, but the most systematic way is mutation testing [24], [25]. In mutation testing bugs are seeded into code as small changes syntactic changes, or mutations. This can be done automatically by defining mutation operators that systematically insert bugs in the code wherever applicable. A mutation score is the corresponding adequacy metric that calculates the test cases' ability to identify (kill) those seeded faults (mutants). For example, if 10 mutations were applied to the software under test, and the test cases reveal (kill) 5 mutants, the mutation score is 50%.

Although covering all the code or killing all mutants still does not guarantee bug-free software, these two metrics, so far, are the best evaluation methods in software testing, which target evaluating the completeness and effectiveness of the generated test cases/oracles. These properties of tests can only be assessed with dynamic execution and thus cannot be evaluated from generic static/textual similarity metrics, which assess only the syntactic similarity (and not semantic or functionality). One simple example of syntactic vs. semantic similarity is when two oracles can be syntactically very similar (e.g., only one character difference; one testing a method with 0 and one with 1, as one of the method's parameter values), but semantically very different (e.g., the zero case covers a very different path in the control flow graph that results to a failure). Thus a purely syntactic metric will have a hard time distinguishing between the two oracles and if used as the only metric for NOG, result in creating sub-optimal oracles with high BLEU values. Therefore, at best, most NOG literature that only evaluates textual similarity metrics implicitly assumes a strong positive correlation between dynamic test adequacy metrics and textual similarity metrics. However, there is no study to support this assumption.

3 Experimental Design

In this section, we explain the neural oracle generation models, our dataset, the evaluation metrics, the correlation analysis used in the paper, our research questions, and their corresponding procedures.

3.1 Neural Oracle Generators (NOGs)

We select three state-of-the-art (SOTA) NOG models based on the following criteria:

- Should be a neural oracle generation model proposing a novel approach for oracle generations.
- Should be published within 3 years to represent SOTA.
- The replication package should be publicly available.
- The study should report evaluation results to their generated oracles' quality (using any comparable metric).

As a result, three NOG models were selected from 13 candidate papers published on main Software Engineering and testing venues (ICSE, FSE, ASE, ISSTA, ICST, AST, TSE, TOSEM, EMSE, IST, and arXiv). Out of 13 candidates, three baselines were selected as 1) two papers did not have a public replication package [8], [26], 2) two used oracle generation as a downstream task on a pre-training model (not a novel approach for oracle generation) [27], [28], 3) four focused on a different task, i.e., test completion [29], bug reproduction [30], enhancing SBST with LLM [31], and whole unit test case generation [32], and 4) two evaluated with a different perspective, i.e. oracle ranking [33] and predicting passing/failing oracles [34]. We included *gpt-3.5* as a baseline for SOTA LLM, which has not been applied on NOG at the time of conducting this study but is deemed relevant. The four selected NOGs in this study are the following:

- **ATLAS** [5]: is the first study to apply a deep neural generation model on oracle generation. It exploits a sequence-to-sequence encoder-decoder (RNN) model to learn and automatically generate an oracle when the test prefix and the focal method are given as input.

- **IR** [6]: extends the previous study *ATLAS* by leveraging information retrieval techniques and integrating deep learning to enhance performance. They also expanded the evaluation by adding unknown vocabulary to make the oracle generation problem more challenging.
- **TOGA** [7]: proposed a unified transformer-based neural model for exceptional and assertion oracles. They first identify if the test method should raise an exception or generate an assertion. If it decides to raise an exception, it generates a try-catch clause. If not, it generates candidate assertions with different parameters. Then an assertion oracle ranker decides the best candidate for generating the assertion.
- **gpt-3.5** [35]: is a generative large language model (LLM) developed by OpenAI. *gpt-3.5* is one of the most widely used LLMs. We wanted to investigate the impact of LLM as it has shown great potential in generating source code for different software engineering tasks [4], [9], [17], [36].

3.2 Dataset

As all three NOG models evaluate their performance with the ATLAS dataset [5], i.e., a set of Java projects with test cases, we reuse the same dataset to facilitate a direct comparison between their results and ours. We do not investigate additional datasets as we deem this dataset comprehensive, i.e. covers over 9K projects in various domains with more than 320K instances. The dataset is a parallel corpus of test prefixes plus the focal method paired with the oracle. However, to calculate the test adequacy metrics, we have to execute each test method and its corresponding focal method. To execute them, we need the package information of the projects that they were originally mined from. In the replication package of the *ATLAS* paper, they provide the resulting raw dataset of test-oracle pairs and the project lists they used for mining. Since the script that constructs the original dataset was not publicly available and each instance was randomly shuffled, retrieving the package information was not possible. So, we replicate and re-implement the *ATLAS* dataset by following the steps reported in their papers.

The steps to mine the *ATLAS* dataset are as follows: First, we clone the project list that is publicly available in the original *ATLAS* paper. We extract all methods with the $@Test$ annotation, which is inherently used by the JUnit framework for unit test cases. For retrieving the oracle, we parse the test methods and look for the specific invocation of assertion APIs, e.g., assertEquals, assertTrue, assertNull, etc. To parse the corresponding focal method, we first extract all methods declared within the project. Then we apply a heuristic to iterate all the invoked methods within the test method. The invoked methods are queried to the list of declared methods in the project. Then we get the last matched method before the oracle statement, assuming it's the focal method. Note that if there is a method invoked within the assertion parameter, we take the method as a focal method instead. Test methods with more than one assert line are filtered as *ATLAS* only focuses on single-line assertion generation as well as duplicate instances.

After retrieving the test-oracle pairs, we split the train, valid, and test set at a project level so that instances in the same project are not in different splits to remove potential information leakage within the projects. We randomly shuffle the projects and assign them to the train-valid-test set with a ratio of 8:1:1, which is the same ratio as the *ATLAS* dataset. For evaluating projects on an execution level, we use a subset of the test set comprised of 13 projects as all the projects weren't deployable. We considered the raw version of *ATLAS* where we keep identifier names since it is a more challenging problem and *IR* and *TOGA* only consider them as well. We also adapt the ideas of *IR*'s new dataset that considers unknown vocabularies to better reflect the data distribution of the real world [6]. We didn't consider *Methods2Test* used in *TOGA* as we only focus on assertion oracles rather than exceptional oracles or whole unit test case generations. As shown in Table 2, after re-implementing the data construction we resulted in a total of 323,994 test-oracle pairs, 261,794 for the train set, 32,476 for validation, and 29,724 for the test set.

TABLE 2: Experiment data

	Training	Validation	Test	Total
#of instances	261.7K	32.4K	29.7K	323.9K
Avg. input tokens	78.96	78.39	85.42	79.99
Avg. output tokens	13	12.87	13.03	12.99
Unique input tokens	706.5K	110.7K	96.4K	851K
Unique output tokens	172.7K	29.2K	22.9K	204.6K

3.3 Evaluation Metrics

3.3.1 Textual Similarity Metrics

Existing oracle generation methods used textual similarity metrics which were originally adopted from the natural language processing (NLP) field to assess the performance of their models. The three NOGs used BLEU and Accuracy to assess their models. However, researchers pointed out that BLEU and Accuracy are not suitable to assess code generation models due to the distinct difference between natural language and source code [37], [38]. For a more comprehensive experiment, we have also added other metrics, i.e., CodeBLEU, METEOR, ROUGE-L, Edit Similarity, and ChrF which are widely used to assess code generation models. Note that all these metrics are textual similarity metrics that compute a similarity score between the generated oracle and the ground-truth reference. All the studied metrics range between 0 to 1, with 0 being a complete mismatch and 1 being a perfect match. However, we report them as rates with percentage points (0 to 100%).

1) **BLEU-4 (abbr. BLEU)** [10] is a widely used evaluation metric that evaluates the text quality generated by the model. It is calculated by an n-gram precision which is the number of matching n-grams from the generated and the ground truth text. We use 4-gram, i.e., BLEU-4, as it is the most widely used and also used by the baseline approaches in this work.
2) **Accuracy (abbr. ACC)** [11] is the number of true predictions over the total amount of instances. The generated is a true positive only if it is an equivalent text, i.e. exact match, to the ground truth.
3) **CodeBLEU (abbr. CB)** [37] is an evaluation metric specifically designed for code generation models. It is calculated by the combination of a weighted n-gram

precision (token similarity), AST matching (syntax similarity), and data flow matching (semantic similarity).

4) **ROUGE-L (abbr. RL)** [39] is another metric used in similarity-based metric. Out of the different ROUGE values, we report the ROUGE-L value which exploits the Longest Common Sub-sequence (LCS) in evaluating the similarity of the texts.
5) **METEOR (abbr. MT)** [40] is also commonly used to evaluate textual similarity. The metric is calculated by the harmonic mean of unigram precision and recall but with a higher weight on recall.
6) **Edit Similarity (abbr. ES)** [41] is a metric that implements Levenshtein distance to measure the distance between two sequences by calculating the smallest number of edit operations (i.e., insertion, deletion, and replacement) required to transform one string into another.
7) **Character n-gram F-score (abbr. ChrF)** [42] focuses on character-level n-grams, i.e., sub-sequence of characters. It is calculated by the harmonic mean of how many n-grams are correct (Precision) and how many reference n-grams are covered (Recall).

We used a Python script to calculate all the textual similarity metrics. We calculated BLEU and METEOR using the `nltk` package. We used (0.25, 0.25, 0.25, 0.25) as the weights for BLEU and a basic smoothing function to match previous studies. For ACC and ChrF, we created a simple script without any packages. For CodeBLEU, we used the script provided by CodeXGLUE [43]. We used the `rouge_scorer` package to calculate ROUGE-L. For Edit similarity, we used `SequenceMatcher` in `difflib` package. All the non-mentioned parameters were set to default.

3.3.2 Test Adequacy Metrics

Following existing work, we use line coverage and mutation score in this work to measure the test adequacy of generated test oracles [44]–[46]. Note that these metrics are calculated by executing the project (thus dynamic) to evaluate the adequacy of the generated tests. All range between 0 (not adequate at all) and 1 (perfect adequacy). We also report them as percentage points (0 to 100%).

1) **Line Coverage**: is a common coverage metric to measure the number of lines covered when a test case is executed. It shows the test case's strength regarding their completeness in exercising the source code linewise.
2) **Mutation Score**: is a fault-based metric to measure the quality of the generated test case by measuring how it can effectively detect synthetic faults (code mutants). The ability to detect a mutant means that the tests are likely to be effective at catching real faults.

3.4 Correlation Analysis

To measure the correlation between each pair of metrics (i.e. one textual similarity and one test adequacy metric), we have chosen two correlation analysis methods, namely Spearman's and Kendall's rank correlation analysis. We chose these two methods because we need a statistical test that can be applied to non-parametric distributions. Spearman's and Kendall's rank correlation measures the correlation between two ranked variables. They are non-parametric measures of statistical dependence between two variables and work on non-normal distributions. The reason for using non-parametric measures is that we cannot guarantee a normal distribution from the variables where parametric tests assume them to be normally distributed. If parametric tests are applied to non-normally distributed variables, it can lead to inaccurate results. Spearman's calculation is calculated based on deviations from the mean ranks, while Kendall's correlation is calculated based on concordant and discordant pairs of ranks. Previous studies have commonly used these methods for correlation analysis on evaluation metrics, thus we follow their methodology [47]–[51].

3.5 Research Questions

We design experiments to answer the following research questions:

RQ1: How do current NOGs perform based on textual similarity evaluation metrics? In the literature, NOGs are evaluated with two metrics, i.e. Accuracy and BLEU score. We added five other majorly used textual similarity metrics, i.e. CodeBLEU, ROUGE-L, METEOR, Edit Similarity, and ChrF for a more comprehensive empirical study. Thus this RQ will show a thorough comparison among SOTA NOG models regarding seven different textual similarity metrics.

RQ2: How do current NOGs perform based on test adequacy evaluation metrics? RQ1 compares different baselines based on the quality of the generated oracles measured by the textual similarity metrics. RQ2 repeats this experiment with dynamic test adequacy metrics. Since the goal of a NOG is to create an oracle as close as possible to a developer-written oracle, we measure how close the line coverage and mutation score of the generated oracles are to the developer-written oracles. Thus the main motivation behind RQ2 is whether we see the same trends when comparing baselines as RQ1.

RQ3: What is the correlation between textual similarity and test adequacy metrics? The main motivation of RQ3 is to come up with a recommendation for the research community on which static textual similarity metrics (if any) can safely be used as a surrogate measure for expensive adequacy metrics when evaluating their NOGs. A high/low correlation between a textual similarity metric and any test adequacy metric will be used as evidence of whether the textual similarity metric should be used in future research in this domain.

RQ4: What are the reasons behind the correlation in RQ3? To further analyze the root cause of correlations in RQ3, we manually analyze samples with mismatched correlation coefficients. Looking at these examples, we want to find out why some of the generated oracles have high scores in one metric category but low in the other. The goal is to provide justifications, based on the observed patterns, for the recommendations given in RQ3.

TABLE 3: The number of injected oracles generated by each NOG. Ms denotes the number of test methods, Cs denotes the number of affected classes, and Ss denotes the number of affected sub-modules.

	ATLAS			*IR*			*TOGA*			*gpt-3.5*		
project name	**Ms**	**Cs**	**Ss**	**Ms**	**Cs**	**Ss**	**Ms**	**Cs**	**Ss**	**Ms**	**Cs**	**Ss**
activemq-artemis	379	177	21	489	211	21	66	43	13	472	210	21
cayenne	417	204	13	460	224	13	94	65	4	463	217	21
cloudstack	410	198	41	466	223	43	190	80	31	466	218	43
cxf	709	291	60	1091	329	62	66	52	22	690	323	62
drill	295	144	39	386	165	40	73	49	17	349	164	40
hadoop	537	356	41	667	426	41	91	67	20	172	92	2
ignite	573	377	15	214	120	8	158	112	9	655	409	15
itext7	520	242	10	606	266	10	61	39	9	586	257	10
jackrabbit-oak	907	504	29	1107	581	29	398	246	21	1111	566	29
james-project	252	121	45	325	138	48	4	4	1	282	129	46
jena	815	273	24	920	293	24	132	49	12	930	293	24
nifi	468	289	118	541	338	128	159	115	64	546	328	128
openmrs-core	507	157	3	540	170	3	215	82	3	391	132	1
total	6789	3333	459	7812	3484	470	1707	1003	226	7113	3338	442

3.6 Experiment Procedure

Experiment Setting for RQ1: We assess the textual similarity evaluation metrics, i.e. BLEU, Accuracy, CodeBLEU, ROUGE-L, METEOR, Edit Similarity, and ChrF on the studied NOGs. To generate oracles from the existing NOGs, i.e., *ATLAS*, *IR*, and *TOGA*, we download the publicly available replication package shared in the paper and follow their instruction to train and run these tools on our dataset. We also report the BLEU and Accuracy scores reported in the original paper for comparison.

For *gpt-3.5*, following existing work [9], we feed a single basic query (zero-shot) to suggest a single line oracle be replaced in the oracle placeholder given the text prefix and its focal method. We used *gpt-3.5* APIs for the model version as it was the latest model accessible to us during the time of our experiment.

We expect there would be a noticeable gap in the textual similarity metrics between our evaluation and the originally reported evaluation as we have divided the train-valid-test splits so that instances from the same project would fall into the same split. This is to mitigate any possible project-specific information leak between the train and the evaluation sets [4].

Experiment Setting for RQ2: To calculate the test adequacy metrics, we build and execute each project and run the test cases with the generated oracles injected into the project. Since the *ATLAS* dataset only considers single-line oracles, there are numerous test cases in the repository that are not our target. So we construct a sub-project for each project that only has the target test cases (those with single-line oracles). Specifically, we first remove all the test cases that are not in our test set for each project. After we get the sub-projects, we replace the generated oracles with the original oracles written by the developers. Since the generated oracles are not guaranteed to be executable, after the replacement, we check if the modified test case with the generated oracles is still executable on the project (note that the original developer-written tests are all executable, before replacing their oracles). We exclude test cases if they become not executable after the oracle injection as we need all test suites to be passing to calculate the test adequacy metrics. The resulting numbers of injected oracles are organized in Table 3. We use PIT [52] to calculate the test adequacy metrics, i.e. line coverage and mutation score. We also calculate and report (as the original score plus/minus the difference after the replacement) the test adequacy metrics per sub-project within the dataset.

Experiment Setting for RQ3: We conduct a correlation analysis between the textual similarity metrics and the test adequacy metrics. We get both metrics at the project level and perform correlation analysis using Spearman's and Kendall's rank correlation coefficient as they are the most common analysis done by different previous metric analysis studies.

The two variables being measured for rankings are one textual similarity metric versus one test adequacy metric. Note that for the adequacy metrics, similar to RQ2, we use the differences (deltas) between the score of the developer-written oracle and the generated oracle. We calculate the correlations between each textual similarity metric and the normalized inverse of $|\text{delta}|$ of line coverage and mutation scores (min-max normalization). We use delta and not the actual metric score to capture the difference in test adequacy between the ground truth (human-written) and the model-generated oracle. This is to match what the similarity metrics are also assessing, capturing the similarity of the ground truth and the model generated. We take the normalized inverse to make the delta (difference/distance) into similarity to match textual similarity metrics.

Since we have seven textual similarity metrics and two test adequacy metrics, we have a total of 14 correlation test runs, per project and baseline. However, the number of samples (projects) per baseline is limited (only 13 projects) which harms the statistical tests p-values. We report this individual baseline analysis in the supplementary material in the replication package. But in the paper, we aggregate the four baseline data and look at a pool of 52 (13 X 4) samples per distribution (i.e., each distribution consists of 52 sample values for a given metric). Then we report the correlation coefficients (ρ or τ) and p-values per correlation test when comparing two metrics (each with a distribution of 52 samples).

From the results, we want to see how many metrics have a significant correlation between them, i.e. p-values of less than 0.05, and ρ or τ values of at least 0.5. We expect these two metrics to have a negative correlation because high similarity metrics mean the two sequences are similar whereas similar code will result in lower $|\text{delta}|$ (difference).

TABLE 4: Performance of NOGs with our new dataset. Each column denotes the score of the seven textual similarity metrics. Bold denotes the best score from a NOG.

NOGs	BLEU	ACC	CB	RL	MT	ES	ChrF
ATLAS	32.88	2.15	12.83	22.21	41.98	46.93	22.77
IR	34.35	5.28	21.29	21.99	40.11	49.17	24.48
TOGA	12.78	9.73	12.79	12.3	12.73	12.93	11.29
gpt-3.5	**49.65**	**11.35**	**26.63**	**44.13**	**51.26**	**61.78**	**30.85**

TABLE 5: Automatic similarity metric scores on *ATLAS*.

Project Name	BLEU	ACC	CB	RL	MT	ES	ChrF
activemq-artemis	34.32	3.66	13.85	20.94	42.41	45.02	17.92
cayenne	**40.66**	0.72	17.76	31.24	**50.42**	**54.06**	28.80
cloudstack	30.42	0.24	11.43	26.22	46.51	50.10	25.88
cxf	34.20	0.14	10.19	25.05	47.17	48.64	24.07
drill	32.95	0.00	10.92	20.20	39.73	43.98	23.20
hadoop	31.85	0.74	12.26	22.39	41.51	47.95	23.40
ignite	36.81	1.05	18.29	28.25	46.91	51.88	25.75
itext7	33.52	**8.46**	13.34	22.90	41.65	47.03	25.70
jackrabbit-oak	30.40	0.33	13.25	17.06	41.84	45.61	19.93
james-project	26.29	0.00	**20.61**	9.35	27.53	36.53	12.59
jena	35.46	0.86	7.61	**32.67**	47.28	50.89	**29.45**
nifi	32.87	2.35	14.72	28.20	49.86	50.15	27.24
openmrs-core	30.11	0.99	8.66	14.20	40.36	42.68	17.46
average	33.07	1.50	13.30	22.97	43.32	47.27	23.18
stdev	3.54	2.33	3.81	6.72	5.96	4.60	4.93

TABLE 6: Automatic similarity metric scores on *IR*.

Project Name	BLEU	ACC	CB	RL	MT	ES	ChrF
activemq-artemis	28.40	3.89	13.16	18.34	35.86	47.14	22.42
cayenne	35.89	0.00	11.72	34.17	41.24	52.54	26.61
cloudstack	44.83	18.24	27.18	34.17	48.51	60.40	39.70
cxf	32.71	1.19	14.12	19.66	38.44	49.96	25.87
drill	25.17	0.00	11.08	14.89	29.26	44.15	20.04
hadoop	**53.08**	**29.54**	**33.97**	**44.37**	**55.77**	**65.92**	**47.05**
ignite	26.61	0.80	8.68	13.52	33.15	46.34	21.90
itext7	31.79	8.42	18.47	19.73	34.87	49.99	26.79
jackrabbit-oak	32.64	4.52	18.11	18.90	40.11	50.06	24.32
james-project	24.83	0.31	14.17	9.17	29.15	40.25	17.20
jena	34.00	6.65	16.28	38.48	50.59	61.19	45.28
nifi	34.00	6.65	16.28	22.10	41.20	50.69	27.65
openmrs-core	28.64	0.74	12.08	10.05	35.94	44.55	19.48
average	33.28	6.23	16.56	22.89	39.55	51.01	28.02
stdev	7.98	8.65	6.96	11.26	8.04	7.42	9.75

We expect a delta score of zero for a perfectly generated oracle. A higher value of |delta| means the two oracles are semantically different.

Experiment Setting for RQ4: In this RQ, we select 104 random samples of generated oracles to understand what causes the correlations. We are specifically interested in cases that have a high disagreement between the textual similarity and adequacy metrics. We pick random 2 samples from each project generated by each NOG where the models are given the same input, i.e., the same generation problem (2 X 13 X 4 = 104).

While doing the manual analysis, we look for any patterns of syntax and semantics of the test code or the oracle itself that may have caused the disagreement.

4 Experimental Results

4.1 RQ1: Textual Similarity Metrics

The results of the four studied NOGs are organized in Table 4. We show the scores of all seven textual similarity metrics discussed in Section 3.3. We also organize the results reported in the original papers for comparison, in Table 1. In Table 1, the results of *ATLAS* are from evaluating the raw dataset (without normalizing identifier names) which is the version used by the other NOGs, i.e. *IR* and *TOGA*. For the results of *IR*, we report the results from the old and new datasets. *IR Old* is the result of using the same dataset as *ATLAS*, without unknown tokens. *IR New* is the new dataset that keeps instances that were dropped from the old dataset due to the unknown tokens. For *TOGA*, we report the results that used *ATLAS* dataset for their assertion oracle generation. We can see that *TOGA* does not report the BLEU score as the methodology treats this problem as a ranking problem. However, considering the first-ranked candidate assertion as a generation output, we can calculate the BLEU score. So for comparison, we have included them in our results.

TABLE 7: Automatic similarity metric scores on *TOGA*.

Project Name	BLEU	ACC	CB	RL	MT	ES	ChrF
activemq-artemis	9.16	7.36	9.18	8.88	9.14	9.23	7.99
cayenne	11.70	5.70	8.27	10.63	11.51	11.99	11.29
cloudstack	**25.81**	**23.97**	19.34	**25.42**	**25.74**	**25.95**	**24.35**
cxf	5.78	5.38	5.79	5.74	5.78	5.80	5.52
drill	6.38	5.45	4.71	6.27	6.34	6.40	6.34
hadoop	8.04	6.92	8.02	7.79	7.98	8.10	7.66
ignite	16.92	11.57	17.13	16.06	16.91	17.21	14.55
itext7	5.17	4.33	5.19	5.03	5.15	5.22	4.66
jackrabbit-oak	20.72	16.01	**20.78**	20.12	20.65	20.88	17.61
james-project	0.27	0.27	0.20	0.27	0.26	0.27	0.27
jena	8.55	7.08	8.54	8.35	8.54	8.63	7.70
nifi	17.04	12.73	12.56	16.26	16.26	17.24	15.50
openmrs-core	17.11	8.19	17.50	15.60	17.00	17.52	10.24
average	11.74	8.84	10.55	11.26	11.64	11.88	10.28
stdev	7.26	6.05	6.37	7.00	7.19	7.34	6.37

TABLE 8: Automatic similarity metric scores on *gpt-3.5*. Bold denotes the best score. The asterisk denotes the best average metric score across all NOGs.

Project Name	BLEU	ACC	CB	RL	MT	ES	ChrF
activemq-artemis	45.11	9.32	28.66	40.48	48.78	60.44	25.41
cayenne	60.31	18.79	**54.67**	56.54	62.39	72.77	45.22
cloudstack	55.30	18.45	33.36	49.03	57.09	68.75	38.28
cxf	53.28	11.16	31.85	51.39	59.66	67.51	35.82
drill	47.36	6.30	35.58	39.67	45.30	60.21	29.79
hadoop	59.23	26.24	40.36	48.23	55.84	66.89	38.14
ignite	43.28	7.79	32.21	39.08	46.96	59.37	27.46
itext7	49.78	17.58	25.80	46.24	51.49	63.03	33.54
jackrabbit-oak	44.78	7.29	26.04	37.64	49.13	60.96	26.19
james-project	43.68	3.19	26.04	31.78	34.60	53.78	20.29
jena	**69.96**	**47.85**	42.54	**68.07**	**70.76**	**78.65**	**63.26**
nifi	54.03	19.05	33.93	49.83	58.72	67.79	37.48
openmrs-core	58.91	10.79	29.44	48.00	56.61	68.88	35.60
average	**52.69***	**15.68***	**33.88***	**46.61***	**53.64***	**65.31***	**35.11***
stdev	8.03	11.69	8.14	9.34	9.05	6.54	10.81

Using the original *ATLAS* dataset, the best accuracy reported in the original papers is *TOGA*. Since TOGA has a very high score on accuracy, it is very likely that it also achieves the best BLEU score because accuracy is a much more strict evaluation metric to achieve. However, since they do not report the exact BLEU score, it is uncertain. So the NOG that has the best reported BLEU score is generated from *IR Old*.

Comparing what is reported in the original papers in Table 1 and the ones we evaluate in Table 4, we can see that the BLEU and accuracy scores reported from the original papers have a big gap from the results we got from the newly processed data. As mentioned in Section 3.5, one possible reason for this is that the splitting of train-valid-test splits at a project level is impacted by removing information leaked into the new test set [4]. By keeping the instances from the same project in each split, similar code structures and project-dependent information are removed from the training set, making it harder for the NOG models to generate project-specific oracles.

TABLE 9: The results of test adequacy metrics for each NOG. LC denotes the line coverage metric and MS denotes the mutation score. Column ±% denotes the absolute difference in percentage points. The green color shows an increase after injection and the red shows a decrease after the injection.

	ATLAS				IR				TOGA				gpt-3.5			
project name	LC	±%	MS	±%	LC	±%	MS	±%	LC	±%	MS	±%	LS	±%	MS	±%
activemq-artemis	28.31	-0.02	27.09	0.79	27.97	-0.23	29.69	-0.26	28.58	**0.24**	28.77	**1.65**	22.71	-0.19	21.68	-0.20
cayenne	46.69	0.00	38.47	0.11	44.32	0.00	40.53	-0.49	56.42	0.00	31.59	0.56	45.71	0.00	41.73	0.32
cloudstack	35.76	0.00	34.21	-0.15	35.19	0.00	32.93	-0.20	38.63	0.00	39.18	-0.40	34.39	1.73	33.50	1.88
cxf	29.56	0.00	25.08	0.00	28.60	0.00	24.75	0.00	22.30	0.00	17.72	0.24	33.44	**2.77**	32.25	0.89
drill	49.04	**0.41**	36.68	**1.06**	35.59	0.00	25.17	**0.36**	14.64	0.00	5.49	0.00	43.18	2.63	26.64	2.30
hadoop	21.02	0.00	13.33	-0.08	27.31	0.00	23.53	0.00	16.07	0.00	6.05	0.00	21.02	-0.66	17.23	-3.06
ignite	48.93	0.00	42.95	0.00	33.78	0.00	37.27	0.00	22.86	0.00	25.90	0.00	48.93	0.00	42.95	-0.72
itext7	53.51	0.00	44.96	-0.41	54.01	0.00	48.99	-0.15	38.85	0.00	15.20	0.00	51.96	0.00	46.13	-0.61
jackrabbit-oak	36.10	0.00	33.93	0.85	36.76	0.00	42.67	0.00	28.03	0.00	26.00	0.06	36.36	-0.68	32.70	-0.83
james-project	31.22	0.00	32.77	-2.87	33.94	-2.74	34.54	-1.73	35.60	0.00	28.30	0.00	33.94	0.00	33.07	-1.39
jena	52.75	-0.05	52.27	0.76	**59.94**	0.00	**60.23**	0.00	38.62	0.00	35.43	0.00	59.87	0.52	57.53	**3.23**
nifi	**58.16**	0.00	**58.48**	0.00	58.16	0.00	58.48	0.00	**58.37**	0.00	**53.38**	0.00	**63.41**	0.00	**59.74**	-0.46
openmrs-core	8.25	0.00	4.82	0.00	8.25	0.00	4.82	0.00	8.25	0.00	4.82	0.00	8.25	0.00	4.85	-0.03
median	36.10	0.00	34.21	0.00	35.19	0.00	34.54	0.00	28.58	0.00	26.00	0.00	36.36	0.00	33.07	-0.20
average	38.41	0.03	34.23	0.00	37.22	-0.23	35.66	-0.19	31.32	0.02	24.45	0.16	38.71	0.47	34.62	0.10
stdev	14.64	0.12	14.63	0.98	14.18	0.76	15.14	0.50	15.16	0.07	14.37	0.49	15.75	1.15	15.39	1.66

Overall, in terms of the average scores across the projects, *gpt-3.5* exhibits the best performance among all textual similarity metrics (See average values with an asterisk in Table 8). In Tables 5-8, we also report the textual similarity metrics at a project level to investigate their effectiveness in each project. Each table has a bold score for each metric score that achieves the highest with its model. For instance, *ATLAS* achieves the highest BLEU, METEOR, and Edit Similarity scores when evaluating *cayenne* project, the highest accuracy on *itext7*, the highest CodeBLEU on *james-project*, the highest ROUGE-L and ChrF score on *jena*. For *IR*, all the best score was achieved when evaluating the *hadoop* project. For *TOGA*, it achieves the highest on six metrics when evaluating *cloudstack*, except for CodeBLEU, which is from *jackrabbit-oak*. And lastly, *gpt-3.5* achieves the best on *jena* for six metrics, except for CodeBLEU which achieves on the *cayenne* project. Looking at all these project-level data, *gpt-3.5* has the highest average metric values (marked with an asterisk in Table 8).

Another clear observation is that the results of the baseline NOGs can vary depending on the project and the variation is not consistent across the techniques. To summarize these variations we also report the standard deviation per baseline metric over the 13 projects. The results show that *gpt-3.5* had the highest standard deviation in five out of seven metrics (only the *IR*'s RL and ES had a higher standard deviation than *gpt-3.5*). Despite having the best performance on the textual similarity metrics, it was shown that it didn't have the best reliability in generating a consistent performance across different projects.

From what is originally reported, *TOGA* has the best overall score for textual similarity metrics. From evaluating the newly curated dataset, *gpt-3.5* has the best scores for all seven textual similarity metrics. There is a big drop in textual similarity metrics when we consider project-level for splitting. However, we also found that *gpt-3.5* had the highest standard deviation, showing that it has a less reliable performance in generating oracles in different projects.

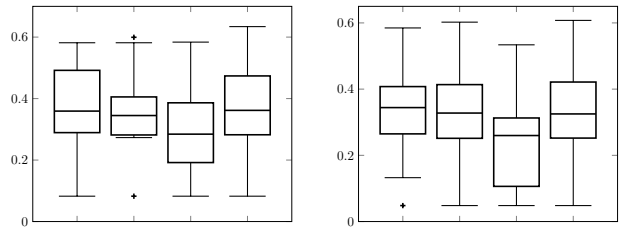
4.2 RQ2: Test Adequacy Metrics

The overall results of test adequacy metrics are shown in Table 9. We also report the box plot of test adequacy metrics to compare the different NOGs evaluated in this study in Fig. 1. As discussed in Section 3.6, we execute the sub-projects before and after replacing the oracles generated by each studied NOG. The test adequacy metrics scores reported in Table 9 are calculated before the injection and the column denoted with ±% on the right side of each metric shows the absolute percentage point difference by calculating the metrics after the generated oracles are injected.

As explained in Section 3.6, the level of increase or decrease in the test adequacy metrics is not big as we are expecting the generated oracle to be similar to the developer-written one. For the results of *ATLAS*, we can see that the highest increase in line coverage and mutation score is from evaluating the *drill* project. For *IR*, *drill* had the highest increase in mutation score but no projects were increased for line coverage. For *TOGA*, *activemq-artemis* had the highest increase in both line coverage and mutation score. For *gpt-3.5*, *cxf* had the highest increase in line coverage and *jena* in mutation score.

To compare which NOG has the best performance overall, we can observe that *gpt-3.5* has the highest increase in line coverage when evaluating *cxf* and mutation score when evaluating *jena*. If we consider the average increase of line coverage, *gpt-3.5* still has the best performance while *TOGA* has the best performance in the average increase of mutation score. Overall, *gpt-3.5* has the highest performance which is in line with the results from RQ1. However, it is also very interesting to find that *gpt-3.5* also has the most number of projects that have a decrease in the test adequacy metrics with 11 out of 26 fields, which accounts for around 42%. This could also tell us that despite having a very high score in textual similarity metrics, it could harm the quality or the strength of the generated oracle. This calls to our attention that we must be careful in using textual similarity metrics as the standard to evaluate NOGs.

Also, unlike what we have observed from textual similarity metrics in RQ1, there was a similar trend in the ranges of scores in test adequacy metrics for projects throughout



(a) Line coverage (b) Mutation score

Fig. 1: Line coverage and mutation scores of *ATLAS*, *IR*, *TOGA*, and *gpt-3.5*, respectively.

the NOGs. For instance, the metric scores for *activemq-artemis* ranged in the 20s, *cloudstack* ranged in the 30s, *james-project* ranged in the 30s, etc. Some projects had higher variance than others, but the patterns and ranges are much clearer than the ones we observed in the results for textual similarity metrics. One reason that we suspect is that the textual similarity metric has a high fluctuation in the score if the model generates even one token that is different from the ground truth text. Since the task is to generate a one-liner assertion oracle with shorter tokens, the metric will have a very big decrease. However for test adequacy, even though the generation is different token-wise, if it correctly generates the same type and values in a similar range for an argument, the executed line of code will be similar. Additionally, *gpt-3.5* showed the highest standard deviation, showing unreliable performance for test adequacy metrics.

Overall, *gpt-3.5* showed the best performance in the test adequacy metrics concerning the maximum score it reaches and the average score. However, we also found that *gpt-3.5* had the most number of projects that have decreased in the test adequacy metrics and the highest standard deviation, showing us that it can be unreliable on different projects. We also found that test adequacy metrics are more stable and have a much clearer trend than textual similarity metrics based on the variation of scores across different projects.

4.3 RQ3: Correlation Analysis

We report the correlation coefficient (ρ and τ) and the p-value of Spearman's and Kendall's rank correlation in Table 10. The bold number in the table shows p-values less than 0.05, which shows the statistical significance of the results. The different colors in the cell show the level of agreement that each coefficient has. Red color shows a very weak agreement, orange color shows a weak agreement, and green color shows a moderate agreement. As expected, the two metrics show a negative correlation, since we compare the inverse delta of test adequacy metrics to the textual similarity metrics. This means the more textually similar oracles (higher score), the lower their semantic differences (lower delta). For ρ, 0.01 to 0.10 is considered a very weak agreement, 0.10 to 0.39 is considered a weak agreement, 0.4 to 0.69 is considered a moderate correlation, 0.70 to 0.89 is considered a strong correlation, and 0.90 to 1.00 is considered a very strong correlation [53]. For τ, 0.01 to 0.24 is considered a very weak agreement, 0.25 to 0.34 is considered a weak agreement, 0.35 to 0.39 is considered a moderate agreement, and 0.40 to 1.0 is considered a strong agreement [54].

TABLE 10: The result of correlation analysis on the two different types of metrics. Bold denotes the p-value less than 0.05 meaning the correlations are statistically significant. The red cell shows "Very Weak Agreement" and the orange cell shows "Weak Agreement". LC and MS are the inverse values of their deltas (details in Section 3.6).

	Spearman's		Kendall's	
Metrics	ρ	**P-value**	τ	**P-value**
LC vs BLEU	-0.3461	**0.0070**	-0.2760	**0.0057**
LC vs ACC	-0.0394	0.3944	-0.0342	0.3773
LC vs CB	-0.2806	**0.0250**	-0.2225	**0.0223**
LC vs RL	-0.3096	**0.0148**	-0.2499	**0.0110**
LC vs MT	-0.3347	**0.0090**	-0.2600	**0.0086**
LC vs ES	-0.2821	**0.0243**	-0.2169	**0.0234**
LC vs ChrF	-0.2234	0.0612	-0.1737	0.0556
MS vs BLEU	-0.3982	**0.0021**	-0.2894	**0.0019**
MS vs ACC	-0.0927	0.2637	-0.0695	0.2444
MS vs CB	-0.3198	**0.0121**	-0.2320	**0.0114**
MS vs RL	-0.3286	**0.0102**	-0.2307	**0.0106**
MS vs MT	-0.3178	**0.0126**	-0.2249	**0.0124**
MS vs ES	-0.2926	**0.0202**	-0.2021	**0.0218**
MS vs ChrF	-0.2567	**0.0371**	-0.1874	**0.0307**

As we can see from the results, except for LC and MS vs Accuracy and LC vs ChrF, the tests from all other pairs result in p-values less than 0.05, which shows their correlation results are statistically significant. However, using either test, all correlations are low. The highest correlation is for MS vs BLEU with ρ = -0.39, which is still considered a weak agreement. Considering these results we can say, that although there is a direct correlation between syntactic and semantic similarity, the strength of the correlations is weak or very weak. Therefore, we do NOT recommend using textual similarity metrics (such as the BLEU category) to evaluate the generated oracle's effectiveness. In other words, unless there is an exact match between the generated oracle and the ground truth, higher BLEU does not necessarily indicate a more effective generated oracle.

Using Spearman's and Kendall's rank correlation analysis, we found that the most closely correlated textual similarity metric to the test adequacy metrics was the BLEU score with $\rho(BLEU vs MS)$=-0.39, which is considered a weak correlation. Based on this finding we recommend NOG researchers avoid only using textual similarity metrics when evaluating NOGs and make sure to include at least one test adequacy metric as their main metric.

4.4 RQ4: Manual Analysis

To further investigate why these two metrics are not significantly correlated, we manually inspect the randomly selected 104 examples of the generated oracles, from all projects. We have found three major types of patterns in the generated oracles. 1) Type-1: identical, 2) Type-2: textually different but same/similar functions, 3) and Type-3: textually similar but different in semantics. Type-1 exhibits a high textual similarity metric and a close to zero

increase/decrease in test adequacy metric. That means both metrics agree that the generated oracle and the developer-written one are similar. Type-2 and 3 are the interesting ones. Type-2 exhibits a low score for textual similarity metrics, however, it achieves close to zero increase/decrease in test adequacy. This means the two oracles are textually different but semantically (from a testing perspective) similar. Type-3 also shows a disagreement between the metrics by exhibiting high textual similarity metrics (textually similar) but different in the testing semantics.

From the 104 samples we inspected, we found 36 Type-1 examples, 37 Type-2 examples, and 1 Type-3 example. *TOGA* had the most Type-1 with 22, *gpt-3.5* had the most Type-2 with 17 and the only Type-3 as well. We give examples of the 3 types for better demonstration in Listing 2. For Type-1, most of the oracles exactly match the ground truth leading to high similarity and identical semantics. As introduced also in Listing 1, the examples demonstrate Type-2 with a high textual difference but similar functions. Oracles with this type tend to have different assertion types to check the status or call different methods in their parameters with similar or identical semantics to the ground truth oracle. An example of Type-3 is also shown where it is textually similar but not semantically. Type-3 has a close textual similarity where the parameters are all included in the ground truth oracle with the BLEU score of 0.54. However, the type of assertion is different and the generated oracle doesn't cover the lines for $IsoMatcher.isomorphic()$. This would affect the test adequacy metrics. The ground truth oracles of this type had multiple or complex method invocations within the parameter, making it hard for the model to generate. These examples show how the generated oracles can cause the two metrics to disagree.

From our manual analysis, we have found three major types of patterns in the generated oracles, and two of them contribute to the nonalignment of textual similarity and test adequacy metrics. Type-1 is an oracle generated with a close agreement to ground truth (high similarity, high test adequacy metric), Type-2 is an oracle with disagreement by low similarity but high test adequacy metric, and Type-3 with disagreement with high similarity but low test adequacy metric. We found that Type-2 is the most common. They tend to have different methods for achieving a similar function. Type-3 wasn't found much however it was due to the model's failure to generate the whole sequence of method chain that hindered the test adequacy metric. The first two types were the cause of major disagreement between the two different metric sets.

4.5 Practical Implication of the Results

From the results of RQ3, we have observed that the textual similarity metrics and test adequacy metrics exhibit "Very Weak Agreement" to "Weak Agreement", meaning they do not have a strong correlation. This empirical evidence implies that using only textual similarity is not sufficient for evaluating NOG models. However, textual similarity to human written tests/oracles is still a good measure of understandability and maintainability [55], [56]. Therefore, our recommendation is to use both types of metrics (textual similarity to human written oracles and test adequacy metrics).

From RQ4, we have observed that the current state-of-the-art models still have challenges in correctly generating oracles with complex parameters (e.g., a long sequence of method chains). To improve the current NOG models, future researchers should: a) guide the models to correctly infer the target library and its APIs and b) use external knowledge to improve oracle generation. These directions could be challenging as library and API information is not provided in the inputs. State-of-the-art LLMs and Retrieval Augmented Generation (RAG) [57] could be a possible solution.

```
// Type-1 example
// Ground truth oracle
assertTrue(answer.getResult())
// Generated oracle from TOGA
assertTrue(answer.getResult())

// Type-2 example
// Ground truth oracle
assertFalse (newMember.isActive(dateToTest))
// Generated oracle
assertEquals (false, newMember.isActive())

// Type-3 example
// Ground truth oracle
assertTrue(IsoMatcher.isomorphic(dsgData,
     dataset.asDatasetGraph()))
// Generated oracle from gpt-3.5
assertEquals(dsgData, dataset.asDatasetGraph())
```

Listing 2: Example of Type-1/2/3.

5 Threats to Validity

Internal Validity. To avoid all confounding factors, we use metrics used in the literature for textual similarity and test adequacy metrics. We use the same implementation of the models and the metrics from the original papers to the best of our ability.

Construct Validity. The main threat to the construct validity can be the evaluation metrics we used. The test adequacy metrics used in this study are line coverage and mutation score. Although these metrics are widely used to evaluate the effectiveness of test cases, they may not be highly correlated to finding actual bugs (the ultimate goal of testing). We plan to examine correlations to real bug detection in our future study.

Conclusion Validity. We have conducted two statistical tests and carefully analyzed the statistical significance when reporting correlations, to avoid conclusion validity threats.

External Validity. The main threats to external validity in this study are the limitations to (a) baseline models (b) test adequacy metrics, (c) the dataset size, (d) and programming language. Regarding baselines, we have used the most recently published state-of-the-art performing models of neural oracle generation models that were publicly available in which they propose a novel way of generating assertion statement lines when the test prefix and focal method are given. Including other baselines might change the observations from this study and will be worth exploring when they are proposed in the future. Regarding test adequacy metrics, we use the line coverage and mutation score. It is possible that our findings do not apply to other levels of code coverage, i.e., branch coverage, statement coverage, or

method coverage. In the future, we plan to include more types of code coverage. Regarding the dataset, although we have put much effort into maintaining high-quality data by using real-world test methods contributed by developers on GitHub and using multiple varieties in the domain of projects, it could not be enough to generalize to all data points. This was inevitable as we had limited resources to evaluate projects that needed to build, execute, and run the entire project to get test adequacy metrics which takes a lot of time and computational complexity. However, we deem we have devised a good number of projects from different domains and a good number of instances in test methods, to show significant observations from the used test set. Regarding the scope of programming language, we only investigate `Java` as all of the most relevant NOGs were based on *ATLAS* which is a `Java` dataset. Our findings are limited to `Java` and whether they could be generalized to other programming languages is left for future studies.

6 Related Work

6.1 Traditional Test Oracle Generation

There have been numerous traditional test oracle generation techniques before neural generation models. In this subsection we very briefly mention them. Peter and Parnas [58] proposed a test oracle generator tool that uses relational program specifications or documents to generate expected outputs of tests as tabular expressions. Bousqpent et al. [59] proposed *Lutess*, a framework that automatically constructs test harnesses from various formal descriptions, i.e. software environment constraints, functional and safety-oriented properties, software operation profiles, and software behavior patterns. Shahamiri et al. [60] proposed an automated test oracle framework using I/O relational analysis to generate the output domain, multi-networks oracle for input-to-output domain mapping, and a comparator to adjust the precision of generated oracle by defining the comparison tolerance. Liu and Shin [61] proposed a new method, *V-method* for automatic test case and test oracle generation from model-based formal specifications. They exploit functional scenarios defined in the formal specification, test generation criteria, algorithms, and mechanisms for deriving test oracles.

6.2 Re-evaluating Evaluation Metrics

Since our study is about re-evaluating NOG evaluation metrics, in this subsection, we also cover most related work that re-evaluates evaluation metrics but in domains other than NOG. Recently, there have been numerous studies about revisiting evaluation metrics in the natural language processing field. Mathur et al. [62] did a re-evaluating study on automatic machine translation evaluation metrics and how they correlate with human judgments. They argue that the current methods for evaluating metrics are unreliable because they depend on the choice and quality of the translations. They also propose a new method for comparing different systems based on how well they agree with human judgments, and how to measure the errors of accepting or rejecting systems that are better or worse than others. Roy et al. [48] empirically investigated how well automatic metrics, such as BLEU, METEOR, and ROUGE, can measure the quality of code summaries generated by data-driven methods. They find that small differences in metric scores (less than 2 points) are not reliable indicators of better summaries and that some metrics (METEOR and ChrF) are more consistent with human evaluations than others (corpus BLEU). Liu et al. [33] pointed out three inappropriate settings in existing evaluation methods of TOGA and comprehensively investigated their impacts on evaluating and understanding the bug-finding performance of TOGA.

Some studies re-visited textual similarity metrics in the code generation domain. Takaichi et al. [63] evaluated various NLP metrics for their suitability in assessing code generated by automated techniques, particularly when the code is syntactically incorrect. They conclude that METEOR is the most effective metric for evaluating the ease of modifying generated code that aligns with the natural language-based inputs. Evtikhiev et al. [64] evaluated six metrics, i.e., BLEU, ROUGE-L, METEOR, ChrF, CodeBLEU, and RUBY for code generation models and found that none can emulate human judgment with high certainty for the CoNaLa dataset. They suggest that ChrF is a better fit for evaluating code generation models, but emphasized the need for a new metric that closely agrees with human evaluation. Liguori et al. [65] evaluated various textual similarity metrics for assessing AI-based offensive code generators, comparing them to human judgment to determine their effectiveness in different contexts. They provided insights into the strengths and limitations of these metrics, highlighting the need for more accurate evaluation methods for code correctness. Sikand et al. [66] revisited the metrics used to evaluate generative AI models for code development. They surveyed satisfaction, well-being, performance, activity, communication, and collaboration. Textual similarity metrics fall into the performance aspect. They concluded that there need to be new metrics concerning the maintainability and the security of generated code. Unlike the above studies, in this work, we revisit the performance of three recent neural oracle generation models and *gpt-3.5* on oracle generation with static textual similarity and dynamic test adequacy metrics.

7 Conclusion

This paper conducted an empirical study of existing neural oracle generation models. We first investigated the models' performance on different textual similarity evaluation metrics. We assessed the generated oracles' performance in their test adequacy metrics, i.e. line coverage and mutation score. We performed a quantitative and qualitative analysis of the textual similarity metrics and the test adequacy to find the correlation of these models and find the gaps between what is currently being used and what we should aim to achieve from the study of this field. We found the correlation between textual similarity metrics and the test adequacy metrics was not significant, meaning the currently assessed textual similarity metrics, i.e. BLEU, accuracy, Rouge-L, METEOR, and CodeBLEU, have no significant relationship with test adequacy metrics, which we use to evaluate the effectiveness of test cases. This shows that textual similarity

metrics are not optimal for showing the quality of the generated oracles and that test adequacy metrics should be considered the main evaluation metrics in this field.

ACKNOWLEDGMENTS

This work was partly supported by NSERC and Alberta Innovates.

REFERENCES

[1] P. Liu, X. Zhang, M. Pistoia, Y. Zheng, M. Marques, and L. Zeng, "Automatic text input generation for mobile testing," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 643–653.

[2] L. Saes, "Unit test generation using machine learning," *Universiteit van Amsterdamg*, 2018.

[3] M. Tufano, S. K. Deng, N. Sundaresan, and A. Svyatkovskiy, "Methods2test: A dataset of focal methods mapped to test cases," in *Proceedings of the 19th International Conference on Mining Software Repositories*. ACM, may 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.12776

[4] J. Shin, S. Hashtroudi, H. Hemmati, and S. Wang, "Domain adaptation for deep unit test case generation," *arXiv e-prints*, pp. arXiv–2308, 2023.

[5] C. Watson, M. Tufano, K. Moran, G. Bavota, and D. Poshyvanyk, "On learning meaningful assert statements for unit test cases," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1398–1409.

[6] H. Yu, Y. Lou, K. Sun, D. Ran, T. Xie, D. Hao, Y. Li, G. Li, and Q. Wang, "Automated assertion generation via information retrieval and its integration with deep learning." ICSE, 2022.

[7] E. Dinella, G. Ryan, T. Mytkowicz, and S. K. Lahiri, "Toga: A neural method for test oracle generation," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2130–2141. [Online]. Available: https://doi.org/10.1145/3510003.3510141

[8] M. Tufano, D. Drain, A. Svyatkovskiy, and N. Sundaresan, "Generating accurate assert statements for unit test cases using pretrained transformers," in *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test*, 2022, pp. 54–64.

[9] Z. Yuan, Y. Lou, M. Liu, S. Ding, K. Wang, Y. Chen, and X. Peng, "No more manual tests? evaluating and improving chatgpt for unit test generation," 2023.

[10] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[11] "Accuracy (trueness and precision) of measurement methods and results — part 1: General principles and definitions," International Organization for Standardization, Geneva, CH, Standard, 1994.

[12] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 416–419.

[13] C. Pacheco and M. D. Ernst, "Randoop: feedback-directed random testing for java," in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, 2007, pp. 815–816.

[14] P. S. Kochhar, F. Thung, and D. Lo, "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 560–564.

[15] R. Baker and I. Habli, "An empirical evaluation of mutation testing for improving the test quality of safety-critical software," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 787–805, 2012.

[16] M. Barani, Y. Labiche, and A. Rollet, "On factors that impact the relationship between code coverage and test suite effectiveness: a survey," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2023, pp. 381–388.

[17] J. Shin, C. Tang, T. Mohati, M. Nayebi, S. Wang, and H. Hemmati, "Prompt engineering or fine tuning: An empirical assessment of large language models in automated software engineering tasks," *arXiv preprint arXiv:2310.10508*, 2023.

[18] T. T. Tanimoto, "Elementary mathematical theory of classification and prediction," 1958.

[19] M. Vijaymeena and K. Kavitha, "A survey on similarity measures in text mining," *Machine Learning and Applications: An International Journal*, vol. 3, no. 2, pp. 19–28, 2016.

[20] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[21] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1536–1547.

[22] J. B. Goodenough and S. L. Gerhart, "Toward a theory of test data selection," in *Proceedings of the international conference on Reliable software*, 1975, pp. 493–510.

[23] P. G. Frankl and E. J. Weyuker, "An applicable family of data flow testing criteria," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1483–1498, 1988.

[24] R. A. Silva, S. d. R. S. de Souza, and P. S. L. de Souza, "A systematic review on search-based mutation testing," *Information and Software Technology*, vol. 81, pp. 19–35, 2017.

[25] A. B. Sánchez, J. A. Parejo, S. Segura, A. Durán, and M. Papadakis, "Mutation testing in practice: Insights from open-source software developers," *IEEE Transactions on Software Engineering*, 2024.

[26] R. White and J. Krinke, "Reassert: Deep learning for assert generation," *arXiv preprint arXiv:2011.09784*, 2020.

[27] A. Mastropaolo, S. Scalabrino, N. Cooper, D. N. Palacio, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Studying the usage of text-to-text transfer transformer to support code-related tasks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 336–347.

[28] A. Mastropaolo, N. Cooper, D. N. Palacio, S. Scalabrino, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Using transfer learning for code-related tasks," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1580–1598, 2022.

[29] P. Nie, R. Banerjee, J. J. Li, R. J. Mooney, and M. Gligoric, "Learning deep semantics for test completion," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2111–2123.

[30] S. Kang, J. Yoon, and S. Yoo, "Large language models are few-shot testers: Exploring llm-based general bug reproduction," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2312–2323.

[31] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 919–931.

[32] Z. Yuan, Y. Lou, M. Liu, S. Ding, K. Wang, Y. Chen, and X. Peng, "No more manual tests? evaluating and improving chatgpt for unit test generation," *arXiv preprint arXiv:2305.04207*, 2023.

[33] Z. Liu, K. Liu, X. Xia, and X. Yang, "Towards more realistic evaluation for neural test oracle generation," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 589–600.

[34] A. R. Ibrahimzada, Y. Varli, D. Tekinoglu, and R. Jabbarvand, "Perfect is the enemy of test oracle," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 70–81.

[35] OpenAI. (2023) Chatgpt. [Online]. Available: https://openai.com/chatgpt

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

[37] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma, "Codebleu: a method for automatic evaluation of code synthesis," *arXiv preprint arXiv:2009.10297*, 2020.

[38] N. Tran, H. Tran, S. Nguyen, H. Nguyen, and T. Nguyen, "Does bleu score work for code migration?" in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 165–176.

[39] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[40] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[41] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "Intellicode compose: Code generation using transformer," in *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2020, pp. 1433–1443.

[42] M. Popović, "chrf deconstructed: beta parameters and n-gram weights," in *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, 2016, pp. 499–504.

[43] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *arXiv preprint arXiv:2102.04664*, 2021.

[44] M. M. Tikir and J. K. Hollingsworth, "Efficient instrumentation for code coverage testing," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 4, pp. 86–96, 2002.

[45] C. Pacheco and M. D. Ernst, "Eclat: Automatic generation and classification of test inputs," in *ECOOP 2005*. Springer, 2005, pp. 504–527.

[46] S. Wang, N. Shrestha, A. K. Subburaman, J. Wang, M. Wei, and N. Nagappan, "Automatic unit test generation for machine learning libraries: How far are we?" in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1548–1560.

[47] J. P. Lim and H. Lauw, "Large-scale correlation analysis of automated metrics for topic models," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 13 874–13 898.

[48] D. Roy, S. Fakhoury, and V. Arnaoudova, "Reassessing automatic evaluation metrics for code summarization tasks," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1105–1116.

[49] A. Shimorina, "Human vs automatic metrics: on the importance of correlation design," 2021.

[50] Y. Graham, T. Baldwin, and N. Mathur, "Accurate evaluation of segment-level machine translation metrics," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 1183–1191.

[51] Y. Liu, Y. Zhou, S. Wen, and C. Tang, "A strategy on selecting performance metrics for classifier evaluation," *International Journal of Mobile Computing and Multimedia Communications (IJMCMC)*, vol. 6, no. 4, pp. 20–35, 2014.

[52] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "Pit: a practical mutation testing tool for java," in *Proceedings of the 25th international symposium on software testing and analysis*, 2016, pp. 449–452.

[53] P. Schober, C. Boer, and L. A. Schwarte, "Correlation coefficients: appropriate use and interpretation," *Anesthesia & analgesia*, vol. 126, no. 5, pp. 1763–1768, 2018.

[54] S. S. Smith, "Scope and methods of political science," *Teaching Political Science*, vol. 9, no. 1, pp. 60–62, 1981.

[55] M. Tufano, D. Drain, A. Svyatkovskiy, S. K. Deng, and N. Sundaresan, "Unit test case generation with transformers and focal context," *arXiv preprint arXiv:2009.05617*, 2020.

[56] D. Roy, Z. Zhang, M. Ma, V. Arnaoudova, A. Panichella, S. Panichella, D. Gonzalez, and M. Mirakhorli, "Deeptc-enhancer: Improving the readability of automatically generated tests," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 287–298.

[57] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking large language models in retrieval-augmented generation," 2024.

[58] D. Peters and D. L. Parnas, "Generating a test oracle from program documentation: work in progress," in *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, 1994, pp. 58–65.

[59] L. Du Bousquet, F. Ouabdesselam, J.-L. Richier, and N. Zuanon, "Lutess: a specification-driven testing environment for synchronous software," in *Proceedings of the 21st international conference on Software engineering*, 1999, pp. 267–276.

[60] S. R. Shahamiri, W. M. N. W. Kadir, S. Ibrahim, and S. Z. M. Hashim, "An automated framework for software test oracle," *Information and Software Technology*, vol. 53, no. 7, pp. 774–788, 2011.

[61] S. Liu and S. Nakajima, "Automatic test case and test oracle generation based on functional scenarios in formal specifications for conformance testing," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 691–712, 2020.

[62] N. Mathur, T. Baldwin, and T. Cohn, "Tangled up in bleu: Reevaluating the evaluation of automatic machine translation evaluation metrics," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 4984–4997.

[63] R. Takaichi, Y. Higo, S. Matsumoto, S. Kusumoto, T. Kurabayashi, H. Kirinuki, and H. Tanno, "Are nlp metrics suitable for evaluating generated code?" in *International Conference on Product-Focused Software Process Improvement*. Springer, 2022, pp. 531–537.

[64] M. Evtikhiev, E. Bogomolov, Y. Sokolov, and T. Bryksin, "Out of the bleu: how should we assess quality of the code generation models?" *Journal of Systems and Software*, vol. 203, p. 111741, 2023.

[65] P. Liguori, C. Improta, R. Natella, B. Cukic, and D. Cotroneo, "Who evaluates the evaluators? on automatic metrics for assessing ai-based offensive code generators," *Expert Systems with Applications*, vol. 225, p. 120073, 2023.

[66] S. Sikand, K. K. Phokela, V. S. Sharma, K. Singi, V. Kaulgud, T. Tung, P. Sharma, and A. P. Burden, "How much space do metrics have in genai assisted software development?" in *Proceedings of the 17th Innovations in Software Engineering Conference*, 2024, pp. 1–5.

Jiho Shin is a Ph.D. candidate at York University, Canada. He obtained his M.Sc. and B.Sc. in Handong Global University, South Korea. His research interest is automated software quality assurance using deep learning and large language models. He has investigated various fields including the explainability of defect prediction models, adequacy of unit test generation and ML-related code generation using language models, and various automated software engineering tasks via prompt engineering. More information can be found at: https://sites.google.com/view/jiho-shin/

Dr. Hadi Hemmati (Senior Member, IEEE) is an associate professor at the electrical engineering and computer science department, at York University. He received his PhD from the University of Oslo, Norway. His main research interests are automated software engineering (with a focus on software testing, debugging, and repair), and trustworthy AI (with a focus on robustness and explainability). More information about him can be found at: https://lassonde.yorku.ca/users/hhemmati

Moshi Wei is a Ph.D. candidate at York University in Canada. He obtained his M.Sc. from the University of Waterloo and his B.Sc. from the University of Ottawa. His research interests lie in improving software reliability and usability by applying deep learning and large language models. He has explored various fields, including automated program repair, software defect prediction, testing and verification of neural network models, software API recommendation, and the misuse of Machine Learning APIs. More information about him can be found at: https://moshiii.github.io

Dr. Song Wang is an associate professor at York University, Canada. He obtained his Ph.D. degree from the University of Waterloo, a Master's degree from the Institute of Software Chinese Academy of Sciences, and dual BE degrees from Sichuan University. He worked at the intersection of Software Engineering and Artificial Intelligence. He has more than 60 high-quality publications including TSE, ICSE, TOSEM, FSE, ASE, etc, and is the recipient of four Distinguished/Best Paper Awards. He is currently serving as an Associate Editor of ACM Transactions on Software Engineering (TOSEM). More information about him can be found at: https://www.eecs.yorku.ca/~wangsong/index.html