Automated Assertion Generation via Information Retrieval and Its Integration with Deep Learning

Hao Yu*
School of Software and Microelectronics, Peking University
China
yh0315@pku.edu.cn

Yiling Lou*
Department of Computer Science, Purdue University
USA
lou47@purdue.edu

Ke Sun
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
China
sunke@stu.pku.edu.cn

Dezhi Ran
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
China
dezhiran@pku.edu.cn

Tao Xie†
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
China
taoxie@pku.edu.cn

Dan Hao
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
China
haodan@pku.edu.cn

Ying Li
National Research Center of Software Engineering, Peking University
China
li.ying@pku.edu.cn

Ge Li
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
China
lige@pku.edu.cn

Qianxiang Wang
Huawei Technologies CO., LTD.
China
wangqianxiang@huawei.com

ABSTRACT

Unit testing could be used to validate the correctness of basic units of the software system under test. To reduce manual efforts in conducting unit testing, the research community has contributed with tools that automatically generate unit test cases, including test inputs and test oracles (e.g., assertions). Recently, ATLAS, a deep learning (DL) based approach, was proposed to generate assertions for a unit test based on other already written unit tests. Despite promising, the effectiveness of ATLAS is still limited. To improve the effectiveness, in this work, we make the first attempt to leverage Information Retrieval (IR) in assertion generation and propose an IR-based approach, including the technique of IR-based assertion retrieval and the technique of retrieved-assertion adaptation. In addition, we propose an integration approach to combine our IR-based approach with a DL-based approach (e.g., ATLAS) to further improve the effectiveness. Our experimental results show that our IR-based approach outperforms the state-of-the-art DL-based approach, and integrating our IR-based approach with the DL-based approach can further achieve higher accuracy. Our results convey an important message that information retrieval could be competitive and worthwhile to pursue for software engineering tasks such as assertion generation, and should be seriously considered by the research community given that in recent years deep learning solutions have been over-popularly adopted by the research community for software engineering tasks.

CCS CONCEPTS

• **Software Testing** → **Unit Tests**; • **Information Retrieval**;

KEYWORDS

Unit Testing, Information Retrieval, Test Assertion, Deep Learning

ACM Reference Format:
Hao Yu, Yiling Lou, Ke Sun, Dezhi Ran, Tao Xie, Dan Hao, Ying Li, Ge Li, and Qianxiang Wang. 2022. Automated Assertion Generation via Information Retrieval and Its Integration with Deep Learning. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510149

1 INTRODUCTION

Unit testing validates the correctness of the software system under test by basic units. A basic unit refers to a small and functionally discrete component (e.g., a method or a class) in a software system. Compared with other levels of testing (e.g., integration testing and system testing), unit testing can help detect and diagnose failures more quickly, especially for complex software systems [11]. A typical unit test includes a test input and test oracle (e.g., assertions),

[*] Equal contribution
[†] Corresponding author

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9221-1/22/05…$15.00
https://doi.org/10.1145/3510003.3510149

and the latter serves as the specifications for the desired behavior on the given test input.

To reduce manual efforts in writing unit tests, a large number of tools have been proposed to generate test inputs automatically, falling into three major categories. (1) Random testing, e.g., Randoop [37], which conducts feedback-directed random testing by analyzing collected execution traces. (2) Dynamic symbolic execution, e.g., JBSE [8], which dynamically conducts symbolic execution with techniques designed to deal with programs that operate on complex heap inputs. (3) Search-based testing, e.g., EvoSuite [15], which applies genetic algorithms to generate and optimize test inputs toward satisfying a coverage criterion.

Besides generating test inputs, such tools can automatically generate assertions with two main categories of traditional approaches (which help detect only crashing faults or regression faults and are incapable of detecting non-crashing faults in the current version in the absence of a previous version). (1) Capture and assert [52]. For example, Randoop [37] and EvoSuite [15] create assertions based on capturing and asserting the return values of all non-void-return methods of the method sequence in the generated test input; EvoSuite further reduces these assertions based on mutation testing [22, 57]. (2) Differential testing [13]. For example, DiffGen [43] generates assertions from runs on two different versions of a class by checking the equality/equivalence of method-call return values and receiver object states from the two versions.

To complement the traditional approaches of assertion generation, based on other already written unit tests, ATLAS [49], a deep learning (DL) based approach, takes a test method without any assertion (i.e., test input only) along with its focal method (i.e., the method under test).[1] In particular, the input to ATLAS is a pair: a test method without assertion along with its focal method, including these two methods' respective method names and method bodies. In the rest of this paper, we refer to such pair as *focal-test* and multiple pairs as *focal-tests*. The output to ATLAS is a meaningful assertion corresponding to the focal-test (i.e., one similar to what developers would have written).

In contrast to the traditional approaches, ATLAS offers two major advantages of assertion generation. First, ATLAS can generate meaningful assertions for detecting non-crashing faults in the current version, whereas the traditional approaches typically help detect only crashing/regression faults with generated assertions being unmeaningful. For example, an industrial evaluation [3] of assertions generated by EvoSuite shows that "in manually written tests, the assertions are meaningful and useful unlike the generated ones". Thus, developers can more desirably inspect assertions recommended by ATLAS when the developers are writing unit tests in their IDE [42]. Second, ATLAS can generate assertions for the code under test in the form of (even uncompilable) source code, whereas the traditional approaches require the code under test to be compilable and executable. In other words, ATLAS can be applicable even during in-progress development of the code under test (e.g., during Test-Driven Development [6, 7]).

However, the effectiveness of ATLAS is restricted by two major limitations inherent in DL. First, assertions generated by ATLAS are not explainable due to the un-explainable nature of DL [16, 47]. Because a recommended assertion may not be an actual assertion applicable for developers, the developers need to inspect the recommended assertion. In this process of inspection, providing explanations on why an assertion is recommended is valuable to assist the developers. Second, approaches based on sequence-to-sequence DL models suffer from the exposure bias [36] and disappearance of the gradient [23, 24], resulting in poor effectiveness in generating a long sequence of tokens as an assertion. In particular, we calculate the accuracy of ATLAS in generating assertions of different lengths on the dataset used by ATLAS, and we find that ATLAS performs well (46.3% accuracy) in generating a short assertion (with fewer than 15 tokens) whereas much worse (only 17.9% accuracy) in generating a long assertion (with more than 15 tokens).

To address the preceding limitations, in this paper, we propose an approach based on information retrieval (IR), including the technique of IR-based assertion retrieval and the technique of retrieved-assertion adaptation. The input and output of the technique of IR-based assertion retrieval are the same as those of ATLAS. This technique first retrieves the assertion whose corresponding focal-test is most similar to the given focal-test based on the Jaccard similarity coefficient. To improve the quality of the retrieved assertion, the technique of retrieved-assertion adaptation further adjusts tokens in the retrieved assertion based on the context. Since the focal-test corresponding to the assertion is also attained when using IR to retrieve the assertion, the developers can attain a valuable reference when inspecting the assertion. In addition, since IR is based on similarity, it is less challenging for our approach to generate a long assertion.

Besides the IR-based approach, in this paper, we also propose an integration approach to integrate our IR-based approach with any DL-based approach (e.g., ATLAS) to enable more powerful assertion generation. In particular, our integration approach carefully examines the compatibility between our retrieved assertion and the current focal-test: if the compatibility is higher than the threshold (determined by a validation set), we directly return the retrieved assertion as the final result; otherwise, we further apply the DL-based approach to generate an assertion.

We evaluate our approaches on two datasets, an existing dataset $Data_{old}$ used in ALTAS and a new dataset $Data_{new}$ constructed in our work. While $Data_{old}$ simplifies the assertion generation problem by excluding some challenging cases (i.e., unknown tokens), $Data_{new}$ is $Data_{old}$'s extended version that includes not only the original $Data_{old}$ but also those excluded cases with unknown tokens. Our experimental results show that the accuracy and BLEU score of our IR-based approach are substantially higher than ATLAS: the technique of IR-based assertion retrieval alone already outperforms ALTAS with 4.84% and 16.34% higher accuracy on both datasets, respectively; in addition, the technique of retrieved-assertion adaptation further improves the accuracy by 7.37% and 2.63% on both datasets, respectively. Furthermore, our integration approach, which combines our IR-based approach with the existing DL-based approach ATLAS, achieves the highest accuracy on assertion generation, i.e., 46.54% and 42.20% on both datasets (additional 15.12% and 20.54% over ATLAS), respectively.

[1] When constructing the dataset used to evaluate ATLAS, the ATLAS authors exclude test methods with multiple assertions, while keeping only test methods each with a single assertion; in addition, the ATLAS authors determine the focal method for a test method t as the last non-JUnit-framework-API method invoked in t.

In summary, this paper makes the following main contributions:

- **IR-based approach.** We make the first attempt to leverage IR in assertion generation, including the technique of IR-based assertion retrieval and the technique of retrieved-assertion adaptation.
- **Integration approach.** We propose an integration approach for integrating our IR-based approach and a DL-based approach (e.g., ATLAS) to enable more powerful assertion generation.
- **Evaluations.** We construct a more comprehensive dataset (compared to the dataset used by ATLAS) including more practical and challenging cases in assertion generation, and extensively evaluate our approaches on both the original dataset and the newly-constructed dataset.
- **Results and practical implication.** Our experimental results show that even the technique of IR-based assertion retrieval alone already outperforms the state-of-the-art DL-based approach, and the technique of retrieved-assertion adaptation further improves the accuracy; moreover, integrating our IR-based approach and the DL-based approach could achieve a higher accuracy. Our results convey an important message that *an IR-based approach can be competitive and worthwhile to pursue for software engineering tasks such as assertion generation, and should be seriously considered by the research community given that in recent years DL solutions have been over-popularly adopted by the research community for software engineering tasks.*

The remainder of this paper is organized as follows. Section 2 introduces the background and related work. Section 3 details the proposed IR-based and integration approaches. Sections 4 and 5 describe the experimental setup and experimental results. Section 6 discusses the threats to validity of our work. Section 7 concludes this paper.

2 BACKGROUND AND RELATED WORK

In this section, we first introduce ALTAS. Then, we detail the related work based on IR and the related work of integrating IR and DL.

2.1 DL-based Assertion Generation

With the recent development of DL, approaches are increasingly proposed to utilize advanced DL techniques to tackle software engineering tasks, such as fault diagnosis [29] and fixing [10, 18, 35, 46], code summarization [19–21], and code clone detection [48, 50, 54, 56, 58]. Recently, Watson et al. [49] propose the first DL-based assertion generation approach named ATLAS, which applies Neural Machine Translation (NMT) to generate an assertion for the given focal-test. ATLAS aims at predicting a meaningful assertion to validate the correctness of the focal method. To collect the dataset for training ATLAS, Watson et al. first extract Test-Assert Pairs (TAPs) from JUnit-based projects in Github, where each pair consists of the focal-test and its relevant assertion. The initial TAP set is then preprocessed into two datasets: (i) raw source code, where TAPs are simply tokenized; (ii) abstract code, where TAPs are first tokenized and the uncommon tokens are further represented by their respective belonging type and sequential id. These two datasets are used to evaluate ATLAS, respectively.

2.2 Information Retrieval

With the rapid expansion of human knowledge and information storage, fast and accurate IR from a massive and multi-modal database becomes an urgent need. In recent years, IR techniques have been widely applied in different software engineering tasks, such as feature location [25], code search [41], concept location [40], software reuse [14, 31, 53], and traceability link recovery between software artifacts [5, 30, 32].

IR techniques fetch the object that best matches the given query from the database. Researchers have leveraged various similarity coefficients in IR. For example, Jaccard [45], a widely-used similarity coefficient, measures the similarity between two sets of data based on their overlapping and unique items. The following formulation presents Jaccard's calculation, where X and Y are two sets. The value varies from 0% to 100%, and a higher value indicates a higher similarity.

$$J(X, Y) = |X \cap Y| / |X \cup Y|$$

2.3 Integration of IR and DL

There exists work that leverages IR to boost DL techniques for tasks other than assertion generation. For example, Liu et al. [26] and Parvez et al. [39] propose to incorporate IR into DL for the task of code generation and summarization. In addition, Liu et al. [26] propose to generate a commit message by a novel hybrid approach of conducting IR followed by DL. Our work shares the same general intuition of "combining IR and DL" with the aforementioned work, but we tackle a different problem (i.e., assertion generation) and we propose a novel integration approach. In addition, our contributions also include the novel IR-based assertion generation and adaptation approaches.

3 APPROACH

In this section, we introduce the proposed approaches in this work. Our approaches include an IR-based approach and an integration approach. The IR-based approach contains two techniques (IR-based assertion retrieval and retrieved-assertion adaptation). Section 3.1 first introduces the technique of IR-based assertion retrieval (in short as IR_{ar}). Given a focal-test in the test set, IR_{ar} retrieves the most similar focal-test in the training set and directly adopts its corresponding assertion as the output. Section 3.2 presents the technique of retrieved-assertion adaptation (in short as RA_{adapt}) to automatically modify the retrieved assertion. Section 3.3 further illustrates an integration approach to combine a DL-based approach (e.g., ATLAS) and the proposed IR-based approach.

3.1 IR-based Assertion Retrieval IR_{ar}

The basic idea of IR_{ar} is to retrieve the assertion whose corresponding focal-test has the highest similarity with the given focal-test. More specifically, IR_{ar} first uses javalang [44] to tokenize each focal-test in the training set and test set, and removes duplicated tokens from each focal-test to enable more efficient retrieval; then, given a focal-test in the test set, IR_{ar} calculates its Jaccard similarity coefficient with all the focal-tests in the training set, and retrieves the one with the highest similarity coefficient; finally, the assertion corresponding to the retrieved focal-test is returned as the expected assertion. When there is a tie in similarity calculation, IR_{ar} selects the first one appearing in the training set.

Algorithm 1 IR-based Assertion Retrieval IR_{ar}

```
Input: $Test_f$: the focal-test to generate assertion for.
Input: $Train_F$: all focal-tests in the training set.
Input: $Train_A$: all corresponding assertions in the training set.
Output: $Retrieved_f$: the focal-test retrieved by $IR_{ar}$.
Output: $Retrieved_a$: the assertion retrieved by $IR_{ar}$.
1: $max \leftarrow 0$ , $Retrieved_f \leftarrow$ “”, $Retrieved_a \leftarrow$ “”
2: for $i = 1$ to $len(Train_F)$ do
3:     $jaccard \leftarrow compute(Test_f, Train_F[i])$
4:     if $jaccard > max$ then
5:         $max \leftarrow jaccard$
6:         $Retrieved_f \leftarrow Train_F[i]$
7:         $Retrieved_a \leftarrow Train_A[i]$
8: return $Retrieved_f, Retrieved_a$
```

Algorithm 1 presents the detailed workflow of IR_{ar}. Being the inputs to the algorithm, $Test_f$ is the input focal-test whose assertion would be generated by IR_{ar}; $Train_F$ and $Train_A$ are the lists of all the focal-tests and their assertions in the training set. IR_{ar} calculates the Jaccard similarity between the input focal-test and each focal-test in the training set (Line 3), selects the one with the highest similarity (Lines 4-7), and returns its assertion as the output (Line 8).

Listing 1: A TAP example in the test set

```
//focal-test:
testReportErrorOnWrongDateEffective(){
  java.lang.String drl="rule␣X␣date-effective␣\"9-asbrdfh-1974\"␣
       when\n" + (("\$s:␣String()"+"then\n") + "end\n");
  org.kie.internal.builder.KnowledgeBuilder kb = org.kie.internal.
       builder.KnowledgeBuilderFactory.newKnowledgeBuilder();
  kb.add(new org.drools.core.io.impl.ByteArrayResource(drl.
       getBytes()), ResourceType.DRL);
}
hasErrors(){
  return (errors) != null;
}
//Assertion to generate:
org.junit.Assert.assertTrue(kb.hasErrors())
```

Listing 2: A TAP example in the training set

```
//focal-test:
testFailedStaticImport(){
  java.lang.String drl="package␣org.drools.test;␣\n" + (((((("" +
       "import␣function␣org.does.not.exist.Foo;␣\n") + "") + "") +
        "rule␣X␣when\n") + "then\n" ) + "end");
  org.kie.internal.builder.KnowledgeBuilder kb = org.kie.internal.
       builder.KnowledgeBuilderFactory.newKnowledgeBuilder();
  kb.add(new org.drools.core.io.impl.ByteArrayResource(drl.
       getBytes()), ResourceType.DRL) ;
}
hasErrors(){
  return (errors) != null;
}
//Assertion:
org.junit.Assert.assertTrue(kb.hasErrors())
```

For example, to generate an assertion for an input focal-test in Listing 1, IR_{ar} compares it with each focal-test in the training set, finds the most similar one (as shown in Listing 2), and returns its corresponding assertion. IR_{ar} is based on the intuition that similar focal-tests are likely to have the identical assertions. It is the reason why IR_{ar} works well for this example. However, it is also very common for two similar focal-tests to share similar (but not identical) assertions. For those cases, directly returning the retrieved assertion would fail to generate a precise assertion. To address this issue, in the next section, we then propose an adaptation technique to modify the retrieved assertion.

3.2 Retrieved-Assertion Adaptation RA_{adapt}

Although IR_{ar} cannot always retrieve fully correct assertions from the training set, it can return “almost correct” assertions that are very similar to the correct ones. Therefore, we further propose an automated adaptation technique RA_{adapt}, which modifies a retrieved assertion toward the correct one based on context information. For a retrieved assertion, RA_{adapt} performs the following adaptation procedure.

- **Step 1: decide whether the assertion should be modified.** Since not all the retrieved assertions need adaptation, a pre-identification should be made first to avoid mistakenly modifying a correct assertion. RA_{adapt} considers an assertion necessary for adaptation if it contains at least one token absent from the input focal-test.
- **Step 2: decide which token (i.e., invoked method, variable, or constant) should be modified.** RA_{adapt} considers a token that is absent from the input focal-test as a *candidate token* for modification.
- **Step 3: decide what value a candidate token should be replaced with.** RA_{adapt} determines the *replacement value* by analyzing the token correspondence between the focal-tests in the test and training sets. We observe that most retrieved assertions have identical syntactic structures as the correct ones but with several inconsistent special tokens (e.g., invoked-method names, variable names, or constant values). Therefore, the current RA_{adapt} mainly considers the replacement operation during adaptation. We believe that RA_{adapt} can be further extended with more edition operations (e.g., addition or deletion) in future work.

Listing 3: A false TAP example in the test set

```
//focal-test:
should_build_an_entity_with_the_right_name(){
  builder.setName("name") ;
  org.bonitasoft.engine.identity.model.SCustomUserInfoDefinition
       entity=builder.done();
}
getName(){
  return name;
}
//Assertion:
org.junit.Assert.assertEquals("name", entity.getName())
```

Listing 4: A false TAP example in the training set

```
//focal-test:
should_build_an_entity_with_the_right_id() {
  builder.setId(1L) ;
  org.bonitasoft.engine.identity.model.SCustomUserInfoDefinition
       entity=builder.done();
}
getId(){
  return id;
}
//Assertion:
org.junit.Assert.assertEquals(1L, entity. getId())
```

Algorithm 2 Retrieved-Assertion Adaptation RA_{adapt}

```
Input: $t$: the focal-test to adapt assertion for.
Input: $t_{ir}$: the focal-test retrieved by $IR_{ar}$.
Input: $a_{ir}$: the assertion retrieved by $IR_{ar}$.
Output: $a'_{ir}$: the assertion adapted by $RA_{adapt}$.
 1: $IM_t, VAR_t, CST_t \leftarrow extractTokens(t)$
 2: $IM_{t_{ir}}, VAR_{t_{ir}}, CST_{t_{ir}} \leftarrow extractTokens(t_{ir})$
 3: $IM_{a_{ir}}, VAR_{a_{ir}}, CST_{a_{ir}} \leftarrow extractTokens(a_{ir})$
 4: $adaptIM \leftarrow needAdapt(IM_t, IM_{t_{ir}}, IM_{a_{ir}})$
 5: $adaptVAR \leftarrow needAdapt(VAR_t, VAR_{t_{ir}}, VAR_{a_{ir}})$
 6: $adaptCST \leftarrow needAdapt(CST_t, CST_{t_{ir}}, CST_{a_{ir}})$
 7: if $!adaptIM$ and $!adaptVAR$ and $!adaptCST$ then
 8:     return $a_{ir}$
 9: $IM'_{t_{ir}}, IM'_t \leftarrow trim(IM_{t_{ir}}, IM_t)$
10: $VAR'_{t_{ir}}, VAR'_t \leftarrow trim(VAR_{t_{ir}}, VAR_t)$
11: $CST'_{t_{ir}}, CST'_t \leftarrow trim(CST_{t_{ir}}, CST_t)$
12: $relpaceIM \leftarrow getReplaceMap(IM'_t, IM'_{t_{ir}}, IM_{a_{ir}})$
13: $relpaceVAR \leftarrow getReplaceMap(VAR'_t, VAR'_{t_{ir}}, VAR_{a_{ir}})$
14: $relpaceCST \leftarrow getReplaceMap(CST'_t, CST'_{t_{ir}}, CST_{a_{ir}})$
15: $a'_{ir} \leftarrow replace(a_{ir}, relpaceIM, relpaceVAR, relpaceCST)$
16: return $a'_{ir}$

17: function $needAdapt(TK_t, TK_{t_{ir}}, TK_{a_{ir}})$
18:     $needAdaptation \leftarrow False$
19:     for $tk_{a_{ir}} \in TK_{a_{ir}}$ do
20:         if $tk_{a_{ir}} \notin TK_t$ and $tk_{a_{ir}} \in TK_{t_{ir}}$ then
21:             $needAdaptation \leftarrow True$
22:             $break$
23:     return $needAdaptation$

24: function $getReplaceMap(TK_t, TK_{t_{ir}}, TK_{a_{ir}})$
25:     $relpaceMap \leftarrow dict()$
26:     for $tk_{t_{ir}} \in (TK_{t_{ir}} \cap TK_{a_{ir}})$ do
27:         $replace \leftarrow$ ""
28:         $max \leftarrow 0$
29:         for $tk_t \in TK_t$ do
30:             $temp \leftarrow computeSimilarity(tk_{t_{ir}}, tk_t)$
31:             if $temp > max$ then
32:                 $max \leftarrow temp$
33:                 $replace \leftarrow tk_t$
34:         $relpaceMap[tk_{t_{ir}}] \leftarrow replace$
35:     return $relpaceMap$
```

Algorithm 2 shows the details of RA_{adapt}. In Lines 1-3, RA_{adapt} first extracts three categories of tokens (all the invoked methods, variables, and constants) defined in t, t_{ir}, and a_{ir} (the inputs to RA_{adapt}), storing these tokens in nine sets. Separately processing the categories of the invoked methods, variables, and constants can make RA_{adapt} more effective because there is no replacement relationship across the three categories. For the example code fragments in Listings 3 and 4, RA_{adapt} extracts the nine sets of tokens: IM_t is [setName, getName, done], VAR_t is [builder, entity, name], and CST_t is ["name"]; $IM_{t_{ir}}$ is [setId, getId, done], $VAR_{t_{ir}}$ is [builder, entity, id], and $CST_{t_{ir}}$ is [1L]; $IM_{a_{ir}}$ is [getId], $VAR_{a_{ir}}$ is [entity], and $CST_{a_{ir}}$ is [1L].

Step 1. In Lines 4-8 and 17-23, RA_{adapt} determines whether an assertion (in the training set) retrieved by IR_{ar} needs to be adapted.

The criterion for the need of adaptation is whether at least an invoked-method in $IM_{a_{ir}}$ has not appeared in IM_t but appeared in $IM_{t_{ir}}$, at least a variable in $VAR_{a_{ir}}$ has not appeared in VAR_t but appeared in $VAR_{t_{ir}}$, or at least a constant in $CST_{a_{ir}}$ has not appeared in CST_t but appeared in $CST_{t_{ir}}$. For the example code fragments in Listings 3 and 4, "getId" in $IM_{a_{ir}}$ has not appeared in IM_t and $1L$ in $CST_{a_{ir}}$ has not appeared in CST_t, so we need to do a replacement analysis for "1L" and "getId".

Step 2. Line 9 of the algorithm removes the overlapping elements between $IM_{t_{ir}}$ and IM_t from each of these two sets, resulting in $IM'_{t_{ir}}$ and IM'_t, respectively. The elements in $IM'_{t_{ir}}$ are the tokens (i.e., invoked methods here) that should be modified. Lines 10 and 11 do the same for the pair of $VAR_{t_{ir}}$ and VAR_t and the pair of $CST_{t_{ir}}$ and CST_t, respectively. For the example code fragments, after the overlapping elements are removed, $IM'_{t_{ir}}$ is [setId, getId], IM'_t is [setName, getName]; $VAR'_{t_{ir}}$ is [id], VAR'_t is [name]; $CST'_{t_{ir}}$ is [1L], CST'_t is ["name"].

Step 3. In Lines 12-14, RA_{adapt} selects all the *replacement values* for invoked methods, variables, and constants; the selection is based on one of the two strategies described in Sections 3.2.1 and 3.2.2. In Lines 15-16, the algorithm replaces the invoked methods, variables, and constants in a_{ir} to be replaced and returns the adapted assertion a'_{ir}.

For Step 3, we design two replacement strategies to decide the replacement value: one is based on heuristics (denoted as RA^{H}_{adapt}) and the other one is based on neural networks (denoted as RA^{NN}_{adapt}). We then introduce these strategies in Sections 3.2.1 and 3.2.2.

3.2.1 Heuristics-based strategy RA^{H}_{adapt}.

In function $getReplaceMap$, $computeSimilarity$ of RA^{H}_{adapt} is calculated based on the sub-tokens resulted from hump splitting of $tk_{t_{ir}}$ in $TK_{t_{ir}}$ and tk_t in TK_t and the positions of $tk_{t_{ir}}$ and tk_t. We first determine whether $tk_{t_{ir}}$ and tk_t contain the same sub-tokens resulted from camel case splitting. If $tk_{t_{ir}}$ and tk_t contain the same sub-tokens after camel case splitting, we put tk_t to a candidate set. Then we select the *replacement value* according to the positions of $tk_{t_{ir}}$ in $TK_{t_{ir}}$ and tk_t in TK_t. Here we select the token tk_t that is closest to the position of token $tk_{t_{ir}}$. For example, "getId" in Listing 4 needs to be respectively compared with "setName" and "getName" in Listing 3. The camel case of "getId" is split into "get" and "Id", the camel case of "getName" is split into "get" and "Name", and the camel case of "setName" is split into "set" and "Name". The position of "getId" in t_{ir} is 34, the position of "getName" in t is 34, and the position of "setName" in t is 7. Therefore, "getId" will be replaced with "getName". The constant $1L$ in Listing 4 needs to be replaced, and $CST'_{t_{ir}}$ has only one element "name", so "1L" is directly replaced with "name".

3.2.2 Neural-network-based strategy RA^{NN}_{adapt}.

The heuristic-based strategy RA^{H}_{adapt} performs replacement based on lexical similarity, which might ignore semantic information embedded in programs. We then propose another neural-network-based replacement strategy RA^{NN}_{adapt}, which augments lexical similarity with semantic information and calculates the replacement value via a neural network architecture for code adaptation. RA_{adapt}^{NN} consists of three components: (1) embedding first-order semantic information, (2) enriching first-order information with high-order semantic information, and (3) combining semantic information with the aforementioned lexical similarity.

In the first component, RA_{adapt}^{NN} captures the first-order information with Word2Vec [1], which maps each word to a n-dimensional vector space $\mathbb{R}$. We use a $m \times n$ matrix M for this semantic mapping, where m denotes the size of vocabulary (a vocabulary typically refers to a set of unique tokens used in the text corpus).

The second component captures high-order information by injecting the semantics of context into the current token. In particular, we employ a bidirectional recurrent neural network architecture (Bi-RNA), which delivers information forward and backward simultaneously. At every step, Bi-RNA updates the representation of the current token I_i with its forward and backward context, enriching I_i's initial first-order information with high-order information. Here R is the recurrent algorithm used to update information, and I_i^R is the output of semantic enhancement of I_i.

$$I_i^R = [R_{>}(I_i, I_{i-1\ldots0}), R_{<}(I_i, I_{i+1\ldots l})]$$

At last, we use a linear layer to combine semantic similarity with the lexical similarity. The semantic similarity is calculated with cosine coefficient, and the cosine coefficient between more similar tokens is larger. In Line 30 of Algorithm 2, we employ dot production to calculate the cosine coefficient between the tokens for replacing $I_i^{Rt_{ir}}$ of $tk_{t_{ir}}$ and its candidate I_i^{Rt} of tk_t (in function *getReplaceMap*) as the following equation. Here S denotes the replacement score of $tk_{t_{ir}}$ and tk_t. $p1$ and $p2$ are the parameters trained by RA_{adapt}^{NN}.

$$S = \begin{cases} p1 \cdot I_i^{Rt_{ir}} \cdot I_i^{Rt} + p2 & I_i^{Rt_{ir}} \text{ and } I_i^{Rt} \text{ contain same sub-token(s)} \\ p1 \cdot I_i^{Rt_{ir}} \cdot I_i^{Rt} & \text{otherwise} \end{cases} \tag{1}$$

RA_{adapt}^{NN} is optimized by maximizing the negative log-likelihood on the replacement possibility S.

3.3 Integration

An IR-based approach retrieves assertions from the training set, and could be highly effective especially when there are similar cases in the training set. On the other hand, a DL-based approach learns to generate assertions based on the training set and is capable of generating "new" assertions absent from the training set. Intuitively, these two approaches are complementary and thus could be further combined to enable more powerful generation of assertions. In this section, we propose an integration approach based on a compatibility inference model, which calculates the "compatibility" between the retrieved assertion and the current focal-test to determine whether to directly return the retrieved assertion or apply a DL-based approach to generate a new assertion.

Figure 1 presents the workflow of our integration approach. The first two steps are our proposed IR-based approach (including the techniques of IR_{ar} and RA_{adapt}), which retrieves an assertion based on the Jaccard similarity and adapts the retrieved assertion if necessary. After adaptation, we then employ a semantic compatibility inference model to calculate the compatibility of the adapted assertion. If the compatibility is lower than the specified threshold (denoted as t in Figure 1), we turn to the DL-based approach (e.g., ATLAS) for generating a "new" assertion. In our approach, t is decided based on the validation set.

We next introduce our compatibility inference model in detail. Recall that the adaptation technique deals with syntactic compatibility by replacing incorrect tokens. We further utilize semantic compatibility to find an incompatible assertion. This problem is similar to the task of neural language inference (NLI) [9], which is usually solved with a binary neural inference model. Typically, a binary neural inference model reasons about evidence between the hypothesis and premise to infer their relationship (i.e., entailment or contradiction). However, our problem is different, where the relationship is "ternary" rather than "binary". Since the assertion is retrieved from its training focal-test, an inference model should not only consider the compatibility between the retrieved assertion and the input focal-test, but also consider the compatibility between the retrieved focal-test and the input focal-test. Based on this insight, we propose a "ternary" neural inference model, which could be viewed as an extension of a recently proposed binary neural inference model named ESIM [9]. We model the local evidence of compatibility among the retrieved assertion, the retrieved focal-test, and the input focal-test with an attention-based RNN model. We then combine the evidence from three code sequences with a neural network, which maps evidence features to the final compatibility.

4 EXPERIMENTAL SETUP

In this section, we describe datasets and metrics used in our experiments. Our source code, datasets, and experimental results are available on our project website [2]. We conduct experiments to answer the following research questions:

- **RQ1:** How does the proposed IR-based assertion retrieval technique IR_{ar} perform compared to the latest DL-based assertion generation approach (i.e., ATLAS)?
- **RQ2:** How do the proposed retrieved-assertion adaptation techniques RA_{adapt}^{H} and RA_{adapt}^{NN} improve the effectiveness?
- **RQ3:** How does the proposed integration approach boost DL-based and IR-based approaches?

4.1 Datasets

4.1.1 $Data_{old}$. We denote the original dataset used by ATLAS as $Data_{old}$, which consists of real-world test assertions from open-source projects in GitHub. As mentioned in previous work [49], $Data_{old}$ is constructed in a simplified way: *it excludes the assertions that contain tokens absent from the focal-test and the vocabulary.* Such tokens are called *unknown tokens* according to the common practice in natural language processing [34, 55]. For example, in Listing 5, the tokens `voter` and `voteMatch` are unknown tokens that do not appear in the focal-test or the vocabulary. $Data_{old}$ excludes such cases, and in total contains 156,760 data items, which are further divided into training, validation, and test sets by the ratio of 8:1:1.

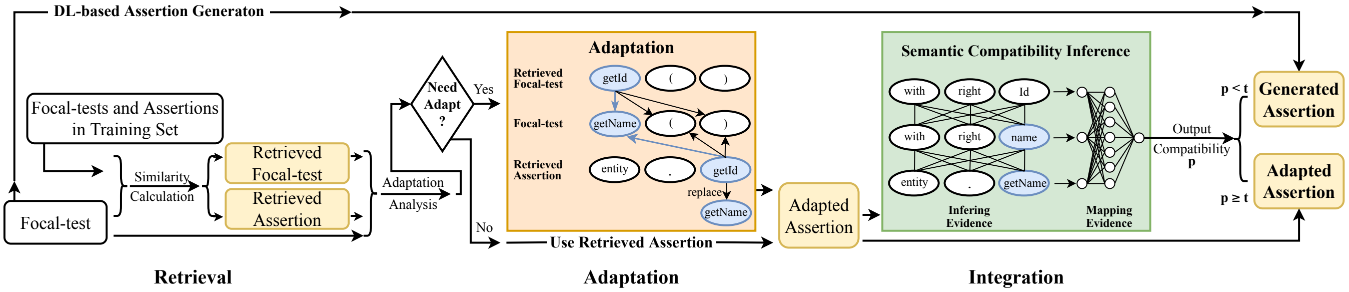


Figure 1: Workflow of the integration approach

4.1.2 *$Data_{new}$.*

In fact, $Data_{old}$ simplifies the test generation problem by excluding some challenging cases (i.e., assertions with unknown tokens) for generation. Therefore, it may fail to represent real-world data distribution and threaten the validity of experimental results. Therefore, in this work, we further construct an extended dataset (i.e., denoted as $Data_{new}$) based on $Data_{old}$ by adding those excluded cases with unknown tokens back to $Data_{old}$. In addition to the original 156,760 items in $Data_{old}$, $Data_{new}$ consists of 108,660 extra items with unknown tokens. In total, the newly constructed dataset $Data_{new}$ contains 265,420 data items, which are also further divided into training, validation, and test sets by the ratio of 8:1:1.

Listing 5: An example of assertion with unknown tokens

```java
//focal-test:
void voteMatch_match_exact_single_sectioned_many_orgs(){
  affiliation.setOrganizationName("philipps␣universitat␣marburg");
  resetOrgNames("some␣other␣inst","philipps␣universitat␣marburg");
}
void resetOrgNames(String... orgNames){
  Mockito.when(getOrgNamesFunction.apply(organization)).thenReturn
        (Lists.newArrayList(orgNames));
}
//Assertion:
assertTrue(voter.voteMatch(affiliation, organization));
```

Table 1 shows the detailed statistics of $Data_{old}$ and $Data_{new}$, including their distribution over different assertion types.

4.2 Metrics

In line with previous work [49], we use the following metrics in our experiments.

4.2.1 *Accuracy.*

We mainly use accuracy to evaluate the effectiveness of assertion generation techniques. In particular, only the assertion that is the same with the ground truth would be considered as accurate. Accuracy calculates the ratio of accurate assertions to all the generated assertions.

4.2.2 *BLEU.*

Following previous work [49], we use the muti-BLEU score to evaluate the similarity of generated assertions with the ground truth. BLEU [38] has been widely used in machine translation systems [4, 19, 21, 27, 33]. The score first calculates the modified n-gram (for BLEU-n, n=1,2,3,4) precision of a candidate sequence (i.e., generated assertions) to the reference message (i.e., the ground truth), and then measures the average modified n-gram precision with a penalty for overly short sentences.

5 EXPERIMENTAL RESULTS

Tables 2 and 3 present the overall effectiveness (i.e., accuracy and BLEU) of all techniques studied in the experiments, including the compared baseline (i.e., ATLAS), IR_{ar}, two adaptation techniques $RA^{H}_{adapt}/RA^{NN}_{adapt}$, and the integration approach.

5.1 RQ1: Effectiveness of IR_{ar}

5.1.1 *Overall effectiveness of IR_{ar}.*

As shown in Tables 2 and 3, the proposed IR_{ar} technique substantially outperforms the compared DL-based approach ATLAS on both datasets in terms of both accuracy and BLEU. In addition, we can observe that the difference is much more prominent on the more challenging dataset (i.e., $Data_{new}$ with unknown tokens). The reason might be that existing DL-based approaches may have a weaker capability of generating assertions with unknown tokens and thus perform substantially worse on $Data_{new}$ (i.e., almost 10% drop in accuracy).

Effectiveness on different assertion types. We further compare the effectiveness of IR_{ar} and ATLAS on assertions of different types. Each column in Table 4 represents the assertion type, and each cell presents the number of correct generated assertions and the corresponding proportions in the brackets. As shown in the table, we observe that IR_{ar} could consistently outperform ATLAS on all the assertion types, indicating the generality of IR_{ar} in generating different assertions.

Effectiveness with different similarity coefficients. IR_{ar} uses Jaccard as its default similarity coefficient. We investigate the impact of different IR similarity coefficients on the effectiveness of IR_{ar}. In particular, we implement two other variants of IR_{ar} with two widely used similarity coefficients, Overlap [51] and Dice [12], which compute the similarity of two given sets (i.e., X and Y) as follows.

$$Overlap(X, Y) = |X \cap Y|/min(|X|, |Y|)$$

$$DSC(X, Y) = |X \cap Y|/(|X| + |Y|)$$

Table 5 presents the accuracy of IR_{ar} with different similarity coefficients on both datasets. The results show that the similarity coefficients have little impact on the effectiveness of IR_{ar}, indicating the generality of the IR_{ar} family in assertion generation.

In summary, our results suggest that the proposed IR_{ar} approach outperforms the existing DL-based approach ATLAS, including on different assertion types and with different IR similarity coefficients. To further understand these results, we next analyze their successful and unsuccessful cases in Sections 5.1.2 and 5.1.3, respectively.

Table 1: Detailed statistics of each type in $Data_{old}$ and $Data_{new}$

AssertType	Total	Equals	True	That	NotNull	False	Null	ArrayEquals	Same	other
$Data_{old}$	15,676	7,866(50%)	2,783(18%)	1,441(9%)	1,162(7%)	1,006(6%)	798(5%)	307(2%)	311(2%)	2(0%)
$Data_{new}$	26,542	12,557(47%)	3,652(14%)	3,532(13%)	1,284(5%)	1071(4%)	735(3%)	362(1%)	319(1%)	3,030(11%)

Table 2: Accuracy of IR_{ar}, RA^{H}_{adapt}, RA^{NN}_{adapt}, and integration

Approach Dataset	ATLAS	IR_{ar}	RA^{H}_{adapt}	RA^{NN}_{adapt}	integration
$Data_{old}$	31.42	36.26	40.97	43.63	46.54
$Data_{new}$	21.66	37.90	39.65	40.53	42.20

Table 3: Muti-BLEU of IR_{ar}, RA^{H}_{adapt}, RA^{NN}_{adapt}, and integration

Approach Dataset	ATLAS	IR_{ar}	RA^{H}_{adapt}	RA^{NN}_{adapt}	integration
$Data_{old}$	68.51	71.48	73.28	73.95	78.86
$Data_{new}$	37.91	57.98	59.81	59.81	60.92

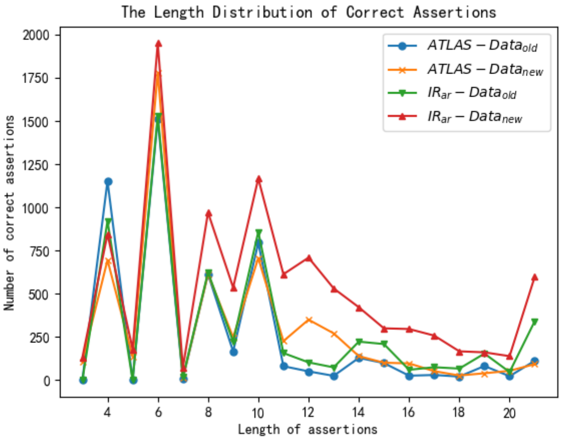


Figure 2: Length distribution of correct assertions

5.1.2 *Correct generated assertions.* To further investigate the correct assertions generated by IR_{ar} and ATLAS, we compare the length of correct assertions (i.e., the number of tokens in each assertion). Figure 2 shows the distribution of the length of correct generated assertions, where the X axis represents the length of assertions and the Y axis represents the number of corresponding assertions. Interestingly, as shown in the figure, we can observe that IR_{ar} tends to outperform ATLAS on longer assertions. For example, on $Data_{old}$, ATLAS is more effective than IR_{ar} when the target assertion contains fewer than eight tokens, whereas ATLAS becomes less effective than IR_{ar} when the target assertion contains more than eight tokens. We also observe similar trends on $Data_{new}$: although IR_{ar} is always more effective than ATLAS, their difference is more prominent with the increasing length of assertions. To further confirm our observation, we then calculate the average lengths of the correct assertions generated by ATLAS and IR_{ar}, respectively. On $Data_{old}$, the average lengths of the correct assertions generated by ATLAS and IR_{ar} are 7.98 and 8.63, respectively; on $Data_{new}$, the average lengths of ATLAS and IR_{ar} are 9.74 and 10.74, respectively. Our results suggest that IR_{ar} has a stronger capability of generating correct long assertions than ATLAS. One potential reason may be that it is still very challenging for existing DL-based approaches to generate long sequences from scratch while it is feasible for IR-based approaches to retrieve long sequences from existing data. In addition, our results also indicate the complementarity between DL-based and IR-based approaches, especially for assertions of different lengths, also motivating the combination of these two approach categories for more powerful assertion generation.

We further look into the correct assertions generated by ATLAS. Interestingly, we find most of them exist in the training set. More specifically, in $Data_{old}$, among 4,925 correct assertions that can be generated by ATLAS, 4,560 (92.59%) assertions exist in the training set if the abstraction strategy is applied. Recall that the abstraction strategy in ATLAS replaces specific tokens (e.g., identifiers and variable names) with abstract tokens to reduce the infrequent tokens and the size of the vocabulary. Similarly, in $Data_{new}$, we also observe that the majority of correct generated assertions (i.e., 98.76%) are the assertions existing in the training set. Intuitively, the strength of DL-based approaches over IR-based approaches lies in intelligently creating new assertions that have never appeared in the training set. However, our results show that most assertions generated by ATLAS exist in the training set while only a few correct assertions are new and "created" by ATLAS. The preceding observation explains why even a simplistic IR approach can outperform a DL-based approach, also suggesting the limited capability of existing DL-based assertion generation in generating correct assertions.

5.1.3 *Incorrect generated assertions.* We then look into the assertions that cannot be successfully generated by ATLAS or IR_{ar}. In particular, we calculate the edit distance between the incorrect generated assertions and the labeled assertions (i.e., the ground truth) in Table 6. Interestingly, although there are a large number of assertions that cannot be successfully generated by ATLAS or IR_{ar}, a considerable ratio of these incorrect generated assertions are very similar to the correct ones, i.e., their edit distance is rather small. For example, among the 10,751 and 20,793 incorrect assertions generated by ATLAS on $Data_{old}$ and $Data_{new}$, respectively, 2,840 (i.e., 26.4%) and 2,641 (i.e., 12.7%) assertions have only one different token from the correct assertions, while 4,517 (i.e., 42.0%) and 5,731 (i.e., 27.6%) assertions have no more than three different tokens from the correct assertions. Similarly, among the 9,992 and 14,483 incorrect

Table 4: Detailed statistics of ATLAS and IR_{ar} for each assert type

AssertType / Approach	Total	Equals	True	That	NotNull	False	Null	ArrayEquals	Same	Other
ATLAS-$Data_{old}$	4,925(31%)	2,501(32%)	966(35%)	248(17%)	598(51%)	229(23%)	236(30%)	100(33%)	47(15%)	0(0%)
IR_{ar}-$Data_{old}$	5,684(36%)	2,957(38%)	1,039(37%)	449(31%)	439(38%)	314(31%)	285(36%)	111(36%)	89(29%)	1(50%)
ATLAS-$Data_{new}$	5,749(22%)	2,900(23%)	619(17%)	537(15%)	388(30%)	126(12%)	85(12%)	47(13%)	37(12%)	1,010(33%)
IR_{ar}-$Data_{new}$	10,059(38%)	4,664(37%)	1,436(39%)	1,070(30%)	600(47%)	394(37%)	286(39%)	147(41%)	113(35%)	1,349(45%)

Table 5: Accuracy of different similarity coefficients

Approach / Dataset	Jaccard	DICE	Overlap
$Data_{old}$	36.26	36.26	36.12
$Data_{new}$	37.90	37.90	37.74

Table 6: Edit distance between correct assertions and incorrect assertions generated by ATLAS and IR_{ar}

Edit / Dataset	1	2	3
ATLAS-$Data_{old}$	2,840(26%)	763(7%)	914(9%)
IR_{ar}-$Data_{old}$	2,966(30%)	1,198(12%)	572(6%)
ATLAS-$Data_{new}$	2,614(13%)	1,595(8%)	1,522(7%)
IR_{ar}-$Data_{new}$	4,532(31%)	1,912(13%)	1,091(8%)

Table 7: Token types to be modified within one edit distance

Token / Dataset	total	api	variable	constant	assertType
ATLAS-$Data_{old}$	2,840	27	865	1,171	738
IR_{ar}-$Data_{old}$	2,966	145	998	1,042	677
ATLAS-$Data_{new}$	2,614	225	482	1,051	274
IR_{ar}-$Data_{new}$	4,532	448	1,014	2,100	395

assertions retrieved by IR_{ar} on $Data_{old}$ and $Data_{new}$, respectively, 2,966 (i.e., 29.7%) and 4,532 (i.e., 31.3%) assertions have only one different token from the correct assertions, while 4,736 (i.e., 47.4%) and 7,535 (i.e., 52.0%) assertions have no more than three different tokens from the correct assertions.

Inspired by the preceding observation, we further categorize those incorrect generated assertions whose edit distance is only one token away from the correct assertions, according to the type of the incorrect token. Table 7 presents the number of assertions in each token category. From the table, we can find that a majority of incorrect assertions generated by ATLAS or IR_{ar} have a wrong constant value compared to the correct assertions, indicating that generating constant values can be a major challenge in assertion generation. Such a challenge is not surprising, since the candidate space of a constant value is often very large, making it challenging to predict a constant value correctly. In addition, we can also find that a considerable number of assertions are incorrect because their assertion types are incorrect. For example, on $Data_{old}$, 738 (677) incorrect assertions generated by ATLAS (IR_{ar}) can be modified into correct assertions if their assertion types are fixed. Compared to incorrect constant values, fixing incorrect assertion types is often less challenging, since fixing types has a limited number of enumerations.

In summary, a considerable number of incorrect assertions generated by ATLAS and IR_{ar} are very similar to correct assertions, and these incorrect assertions can still be helpful to developers or provide useful information for assertion generation. In addition, many incorrect assertions may become correct after only one token (e.g., related to assertion type) is modified. Furthermore, our results also motivate the intuition of our adaptation technique.

5.2 RQ2: Effectiveness of RA_{adapt}

In Table 8, Column "Total" shows the overall accuracy of RA^{H}_{adapt} and RA^{NN}_{adapt} on $Data_{old}$ and $Data_{new}$. We can find that both adaptation techniques can substantially improve IR_{ar}, and RA^{NN}_{adapt} can achieve higher improvement than RA^{H}_{adapt}. For example, RA^{H}_{adapt} and RA^{NN}_{adapt} improve IR_{ar} by 4.71% and 7.37% on $Data_{old}$, while 1.75% and 2.63% on $Data_{new}$, respectively. In addition, RA^{H}_{adapt} (RA^{NN}_{adapt}) can further outperform the DL-based approach ATLAS substantially, i.e., 9.55% (12.21%) improvement on $Data_{old}$ and 17.99% (18.87%) improvement on $Data_{new}$. Furthermore, the other columns in Table 8 present the effectiveness of adaptation on assertions of different types. From the table, we can observe a consistent improvement of both adaptation techniques on all the assertion types.

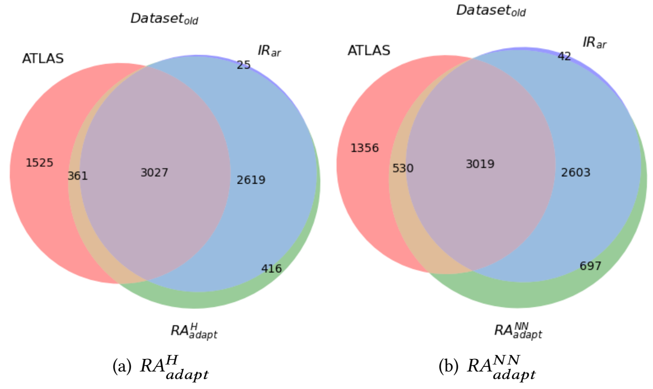
We further analyze the successful and unsuccessful cases of our adaptation techniques. In particular, we find that on $Data_{old}$, 419 and 737 incorrect retrieved assertions are modified into correct assertions by RA^{H}_{adapt} and RA^{NN}_{adapt}, respectively; similarly, on $Data_{new}$, 563 and 679 incorrect retrieved assertions are modified into correct ones by RA^{H}_{adapt} and RA^{NN}_{adapt}, respectively. On the other hand, our adaptation techniques are likely to modify a correct assertion into an incorrect one. On $Data_{old}$, a small number of correct retrieved assertions (i.e., 25 and 42) are mistakenly identified by RA^{H}_{adapt} and RA^{NN}_{adapt}, respectively, as incorrect assertions and further modified into incorrect assertions, and the numbers on $Data_{new}$ are 68 and 84. In summary, our adaptation techniques successfully modify a considerable number of incorrect assertions into correct ones at the cost of mistakenly modifying a small number of correct retrieved assertions, explaining the overall effectiveness of adaptation.

Table 8: Detailed statistics of RA^{H}_{adapt} and RA^{NN}_{adapt} for each assert type

AssertType Approach	Total	Equals	True	That	NotNull	False	Null	ArrayEquals	Same	Other
RA^{H}_{adapt}-$Data_{old}$	6,423(41%)	3,300(42%)	1,151(41%)	536(37%)	553(48%)	335(33%)	316(40%)	120(39%)	111(36%)	1(50%)
RA^{NN}_{adapt}-$Data_{old}$	6,839(44%)	3,509(45%)	1,225(44%)	551(38%)	610(52%)	342(34%)	341(43%)	134(44%)	126(41%)	1(50%)
RA^{H}_{adapt}-$Data_{new}$	10,525(40%)	4,882(39%)	1,487(41%)	1,142(32%)	651(51%)	403(38%)	297(40%)	154(43%)	121(38%)	1,388(46%)
RA^{NN}_{adapt}-$Data_{new}$	10,758(41%)	4,988(40%)	1,526(42%)	1,161(33%)	691(54%)	401(37%)	308(42%)	162(45%)	126(39%)	1,395(46%)

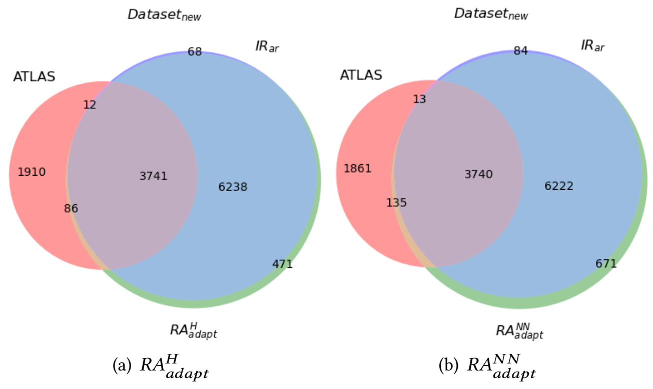
Table 9: Detailed statistics of integration for each assert type

AssertType Approach	Total	Equals	True	That	NotNull	False	Null	ArrayEquals	Same	Other
integration-$Data_{old}$	7,295(47%)	3,714(47%)	1,333(48%)	546(38%)	724(62%)	348(35%)	352(44%)	148(48%)	129(41%)	1(50%)
integration-$Data_{new}$	11,201(42%)	5,248(42%)	1,566(43%)	1,196(34%)	711(55%)	401(37%)	313(43%)	162(45%)	128(40%)	1,476(49%)



(a) RA^{H}_{adapt} (b) RA^{NN}_{adapt}

Figure 3: Complementarity between ATLAS, IR_{ar}, RA^{H}_{adapt}, and RA^{NN}_{adapt} in $Data_{old}$



(a) RA^{H}_{adapt} (b) RA^{NN}_{adapt}

Figure 4: Complementarity between ATLAS, IR_{ar}, RA^{H}_{adapt}, and RA^{NN}_{adapt} in $Data_{new}$

5.3 RQ3: Effectiveness of Integration

5.3.1 Complementarity between DL-based and IR-based approaches. Figures 3 and 4 present the overlapping between the correct assertions generated by the DL-based approach (i.e., ATLAS) and the IR-based approaches (i.e., IR_{ar} and RA_{adapt}). From the figures, we can find that DL-based and IR-based approaches are complementary. For example, on $Data_{old}$, between ATLAS and RA^{H}_{adapt}, 1,525 correct assertions are uniquely generated by ATLAS, while 3,035 correct assertions are uniquely generated by RA^{H}_{adapt}; between ATLAS and RA^{NN}_{adapt}, 1,356 correct assertions are uniquely generated by ATLAS, while 3,300 correct assertions are uniquely generated by RA^{NN}_{adapt}. Such a complementarity is consistent on $Data_{new}$. The results indicate that integrating both approaches can even enable more powerful assertion generation.

5.3.2 Effectiveness of integration. Table 9 presents the effectiveness of integration on all assertions and assertions of each type. From the table, we can observe that integration consistently outperforms DL-based and IR-based approaches on all assertions. The results suggest the effectiveness of our proposed integration approach, also showing that integrating DL-based and IR-based approaches can be a promising direction for automated assertion generation.

6 THREATS TO VALIDITY

One major threat in our work comes from not comparing the ability to generate assertions that are compilable and can find bugs between our approaches and other existing approaches of test case generation. Although being compilable can be another indicator for assertion quality, automated build has always been a challenging task, highly dependent on external/internal settings/resources [17, 28]. Therefore, it is challenging to automatically compile large-scale subjects. To ensure the scale of our experiments, we do not use being compilable and the ability to find bugs as metrics. Note that we check the syntax of assertions generated by our IR-based approach, and all the generated assertions conform to the syntax. These results are expected since the retrieval mechanism of our IR-based approach can ensure the syntactic correctness of the generated assertions.

7 CONCLUSION

In this paper, we have made the first attempt to leverage Information Retrieval (IR) in assertion generation and propose an IR-based approach including the technique of IR_{ar} and the technique of RA_{adapt}. We have also proposed an integration approach for integrating our IR-based approach and a DL-based approach such as ATLAS to further improve the effectiveness. Our experimental results have shown that ATLAS can be outperformed by IR_{ar}, which achieves the accuracy of 36.26% and 37.90% on two datasets, respectively. Moreover, our RA^{H}_{adapt} technique can help achieve accuracy of 43.63% and 40.53% on both datasets, respectively. Finally, the integration approach achieves the accuracy of 46.54% and 42.20% on both datasets. Our work has conveyed an important message that an IR-based approach can be competitive and worthwhile to pursue for software engineering tasks such as assertion generation, and should be seriously considered by the research community given that in recent years DL solutions have been over-popularly adopted by the research community for software engineering tasks.

ACKNOWLEDGMENTS

This work was partially supported by National Natural Science Foundation of China (Grant No. 62161146003, 62072007, 62192733), a grant from Huawei, and the Tencent Foundation/XPLORER PRIZE.

REFERENCES

[1] 2013. Word2vec embeddings. https://radimrehurek.com/gensim/models/word2vec.html.

[2] 2022. https://github.com/yh1105/Artifact-of-Assertion-ICSE22.

[3] M. Moein Almasi, Hadi Hemmati, Gordon Fraser, Andrea Arcuri, and Janis Benefelds. 2017. An Industrial Evaluation of Unit Test Generation: Finding Real Faults in A Financial Application. In *Proceedings of the 39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. 263–272. https://doi.org/10.1109/ICSE-SEIP.2017.27

[4] Fan Angela, Bhosale Shruti, Schwenk Holger, Ma Zhiyi, El-Kishky Ahmed, Goyal Siddharth, Baines Mandeep, Celebi Onur, Wenzek Guillaume, Chaudhary Vishrav, Goyal Naman, Birch Tom, Liptchinsky Vitaliy, Edunov Sergey, Grave Edouard, Auli Michael, and Joulin Armand. 2021. Beyond English-Centric Multilingual Machine Translation. *Journal of Machine Learning Research* 22 (2021), 107:1–107:48.

[5] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. 2002. Recovering Traceability Links Between Code and Documentation. *IEEE Transactions on Software Engineering* 28, 10 (2002), 970–983. https://doi.org/10.1109/TSE.2002.1041053

[6] Dave Astels. 2003. *Test Driven Development: A Practical Guide.* Prentice Hall Professional Technical Reference.

[7] Kent Beck. 2003. *Test-driven Development: By Example.* Addison-Wesley Professional; 1st edition (November 8, 2002).

[8] Pietro Braione, Giovanni Denaro, and Mauro Pezzè. 2016. JBSE: A Symbolic Executor for Java Programs with Complex Heap Inputs. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*. 1018–1022. https://doi.org/10.1145/2950290.2983940

[9] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced LSTM for Natural Language Inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*. 1657–1668. https://doi.org/10.18653/v1/P17-1152

[10] Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2021. SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair. *IEEE Transactions on Software Engineering* 47, 9, 1943–1959. https://doi.org/10.1109/TSE.2019.2940179

[11] Mike Cohn. 2010. *Succeeding with Agile: Software Development Using Scrum (1st ed.).* Addison-Wesley Professional; 1st edition (October 26, 2009).

[12] Lee R. Dice. 1945. Measures of the Amount of Ecologic Association Between Species. *Ecological Society of America* (1945).

[13] Robert B. Evans and Alberto Savoia. 2007. Differential Testing: A New Approach to Change Detection. In *Proceedings of the 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers (ESEC/FSE)*. 549–552. https://doi.org/10.1145/1295014.1295038

[14] William B. Frakes and Kyo Kang. 2005. Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering* 31, 7 (2005), 529–536. https://doi.org/10.1109/TSE.2005.85

[15] Gordon Fraser and Andrea Arcuri. 2011. 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 (ESEC/FSE)*. 416–419. https://doi.org/10.1145/2025113.2025179

[16] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2019. A Survey of Methods for Explaining Black Box Models. *Comput. Surveys* 51, 5 (2019), 93:1–93:42. https://doi.org/10.1145/3236009

[17] Foyzul Hassan and Xiaoyin Wang. 2018. HireBuild: An Automatic Approach to History-Driven Repair of Build Scripts. In *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering (ICSE)*. 1078–1089. https://doi.org/10.1145/3180155.3180181

[18] Hideaki Hata, Emad Shihab, and Graham Neubig. 2018. Learning to Generate Corrective Patches using Neural Machine Translation. *CoRR* abs/1812.07170 (2018).

[19] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep Code Comment Generation. In *Proceedings of the 26th International Conference on Program Comprehension (ICPC)*. 200–210. https://doi.org/10.1145/3196321.3196334

[20] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2020. Deep Code Comment Generation with Hybrid Lexical and Syntactical Information. *Empirical Software Engineering* 25, 3 (2020), 2179–2217. https://doi.org/10.1007/s10664-019-09730-9

[21] Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. 2018. Summarizing Source Code with Transferred API Knowledge. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. https://doi.org/10.24963/ijcai.2018/314

[22] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678. https://doi.org/10.1109/TSE.2010.62

[23] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the Limits of Language Modeling. *CoRR* abs/1602.02410 (2016).

[24] Oleksii Kuchaiev and Boris Ginsburg. 2017. Factorization Tricks for LSTM Networks. In *Workshop Track Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

[25] Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Václav Rajlich. 2007. Feature Location via Information Retrieval Based Filtering of A Single Scenario Execution Trace. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 234–243. https://doi.org/10.1145/1321631.1321667

[26] Shangqing Liu, Yu Chen, Xiaofei Xie, Jing Kai Siow, and Yang Liu. 2020. Retrieval-Augmented Generation for Code Summarization via Hybrid GNN. In *International Conference on Learning Representations (ICLR)*.

[27] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual Denoising Pre-training for Neural Machine Translation. *Transactions of the Association for Computational Linguistics* 8 (2020), 726–742. https://doi.org/10.1162/tacl_a_00343

[28] Yiling Lou, Zhenpeng Chen, Yanbin Cao, Dan Hao, and Lu Zhang. 2020. Understanding Build Issue Resolution in Practice: Symptoms and Fix Patterns. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 617–628. https://doi.org/10.1145/3368089.3409760

[29] Yiling Lou, Qihao Zhu, Jinhao Dong, Xia Li, Zeyu Sun, Dan Hao, Lu Zhang, and Lingming Zhang. 2021. Boosting Coverage-Based Fault Localization via Graph-Based Representation Learning. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 664–676. https://doi.org/10.1145/3468264.3468580

[30] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. 2007. Recovering Traceability Links in Software Artifact Management Systems Using Information Retrieval Methods. *ACM Transactions on Software Engineering and Methodology* 16, 4 (2007), 13–es. https://doi.org/10.1145/1276933.1276934

[31] Yoëlle S. Maarek, Daniel M. Berry, and Gail E. Kaiser. 1991. An Information Retrieval Approach for Automatically Constructing Software Libraries. *IEEE Transactions on Software Engineering* 17, 8 (Aug. 1991), 800–813. https://doi.org/10.1109/32.83915

[32] Andrian Marcus, Jonathan I. Maletic, and Andrey Sergeyev. 2005. Recovery of Traceability Links Between Software Documentation and Source Code. *International Journal of Software Engineering and Knowledge Engineering* 15, 05 (2005), 811–-836. https://doi.org/10.1145/1276933.1276934

[33] Popel Martin, Tomkova Marketa, Tomek Jakub, Kaiser Łukasz, Uszkoreit Jakob, Bojar Ondřej, and Žabokrtský Zdeněk. 2020. Transforming Machine Translation: A Deep Learning System Reaches News Translation Quality Comparable to Human Professionals. *Nature Communications* (2020).

[34] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and Optimizing LSTM Language Models. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.

[35] Ali Mesbah, Andrew Rice, Emily Johnston, Nick Glorioso, and Edward Aftandilian. 2019. DeepDelta: Learning to Repair Compilation Errors. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 925–936. https://doi.org/10.1145/3338906.3340455

[36] Kenton Murray and David Chiang. 2018. Correcting Length Bias in Neural Machine Translation. In *Proceedings of the 3rd Conference on Machine Translation (WMT)*. 212–223. https://doi.org/10.18653/v1/w18-6322

[37] Carlos Pacheco and Michael D. Ernst. 2007. Randoop: Feedback-Directed Random Testing for Java. In *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion (OOPSLA)*. 815–816. https://doi.org/10.1145/1297846.1297902

[38] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. 311–318. https://doi.org/10.3115/1073083.1073135

[39] Md Rizwan Parvez, Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval Augmented Code Generation and Summarization. arXiv:2108.11601

[40] Maksym Petrenko and Václav Rajlich. 2013. Concept Location Using Program Dependencies and Information Retrieval. *Information and Software Technology* 55, 4 (2013), 651–659. https://doi.org/10.1016/j.infsof.2012.09.013

[41] Renuka Sindhgatta. 2006. Using an Information Retrieval System to Retrieve Source Code Samples. In *Proceedings of the 28th International Conference on Software Engineering (ICSE)*. 905–908. https://doi.org/10.1145/1134285.1134448

[42] Yoonki Song, Suresh Thummalapenta, and Tao Xie. 2007. UnitPlus: Assisting Developer Testing in Eclipse. In *Proceedings of the OOPSLA Workshop on Eclipse Technology EXchange*. 26–30. https://doi.org/10.1145/1328279.1328285

[43] Kunal Taneja and Tao Xie. 2008. Automated Regression Unit-Test Generation. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 407–410. https://doi.org/10.1109/ASE.2008.60

[44] Chris Thunes. 2019. Javalang. https://github.com/c2nes/javalang

[45] Tanimoto TT. 1957. An Elementary Mathematical Theory of Classification and Prediction. *Internal IBM Technical Report* (1957).

[46] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2018. An Empirical Investigation Into Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. 832–837. https://doi.org/10.1145/3238147.3240732

[47] Jingyuan Wang, Yufan Wu, Mingxuan Li, Xin Lin, Junjie Wu, and Chao Li. 2020. Interpretability is a Kind of Safety: An Interpreter-based Ensemble for Adversary Defense. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. 15–24. https://doi.org/10.1145/3394486.3403044

[48] Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. 2020. Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree. In *Proceedings of the 27th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 261–271. https://doi.org/10.1109/SANER48275.2020.9054857

[49] Cody Watson, Michele Tufano, Kevin Moran, Gabriele Bavota, and Denys Poshyvanyk. 2020. On Learning Meaningful Assert Statements for Unit Test Cases. In *Proceedings of the 42th IEEE/ACM International Conference on Software Engineering (ICSE)*. 1398–1409. https://doi.org/10.1145/3377811.3380429

[50] Hui-Hui Wei and Ming Li. 2017. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. 3034–3040.

[51] Wikipedia contributors. 2021. Overlap — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Overlap&oldid=1061948530 [Online; accessed 12-March-2022].

[52] Tao Xie. 2006. Augmenting Automatically Generated Unit-Test Suites with Regression Oracle Checking. In *Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP)*. 380–403. https://doi.org/10.1007/11785477_23

[53] Yunwen Ye and Gerhard Fischer. 2002. Supporting Reuse by Delivering Task-Relevant and Personalized Information. In *Proceedings of the 24th International Conference on Software Engineering (ICSE)*. 513–523. https://doi.org/10.1145/581339.581402

[54] Hao Yu, Wing Lam, Long Chen, Ge Li, Tao Xie, and Qianxiang Wang. 2019. Neural Detection of Semantic Code Clones Via Tree-Based Convolution. In *Proceedings of the 27th IEEE/ACM International Conference on Program Comprehension (ICPC)*. 70–80. https://doi.org/10.1109/ICPC.2019.00021

[55] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent Neural Network Regularization. *arXiv preprint arXiv:1409.2329* (2014).

[56] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A Novel Neural Source Code Representation Based on Abstract Syntax Tree. In *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE)*. 783–794. https://doi.org/10.1109/ICSE.2019.00086

[57] Jie Zhang, Lingming Zhang, Mark Harman, Dan Hao, Yue Jia, and Lu Zhang. 2019. Predictive Mutation Testing. *IEEE Transactions on Software Engineering* 45, 9 (2019), 898–918. https://doi.org/10.1109/TSE.2018.2809496

[58] Gang Zhao and Jeff Huang. 2018. DeepSim: Deep Learning Code Functional Similarity. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 141–151. https://doi.org/10.1145/3236024.3236068