Revisiting and Improving Retrieval-Augmented Deep Assertion Generation

Weifeng Sun*, Hongyan Li*, Meng Yan†, Yan Lei, Hongyu Zhang
School of Big Data and Software Engineering, Chongqing University, Chongqing, China
{weifeng.sun, hongyan.li, mengy, yanlei, hyzhang}@cqu.edu.cn

***Abstract*—Unit testing validates the correctness of the unit under test and has become an essential activity in software development process. A unit test consists of a test prefix that drives the unit under test into a particular state, and a test oracle (e.g., assertion), which specifies the behavior in that state. To reduce manual efforts in conducting unit testing, Yu et al. proposed an integrated approach (*integration* for short), combining information retrieval with a deep learning-based approach, to generate assertions for a unit test. Despite being promising, there is still a knowledge gap as to why or where *integration* works or does not work. In this paper, we describe an in-depth analysis of the effectiveness of *integration*. Our analysis shows that: ① The overall performance of *integration* is mainly due to its success in retrieving assertions. ② *integration* struggles to understand the semantic differences between the retrieved focal-test (*focal-test* includes a test prefix and a unit under test) and the input focal-test, resulting in many tokens being incorrectly modified; ③ *integration* is limited to specific types of edit operations (i.e., replacement) and cannot handle token addition or deletion. To improve the effectiveness of assertion generation, this paper proposes a novel retrieve-and-edit approach named EDITAS. Specifically, EDITAS first retrieves a similar focal-test from a pre-defined corpus and treats its assertion as a prototype. Then, EDITAS reuses the information in the prototype and edits the prototype automatically. EDITAS is more generalizable than *integration* because it can ❶ comprehensively understand the semantic differences between input and similar focal-tests; ❷ apply appropriate assertion edit patterns with greater flexibility; and ❸ generate more diverse edit actions than just replacement operations. We conduct experiments on two large-scale datasets and the experimental results demonstrate that EDITAS outperforms the state-of-the-art approaches, with an average improvement of 10.00%-87.48% and 3.30%-42.65% in accuracy and BLEU score, respectively.**

***Index Terms*—Unit Testing, Assertion Generation, Test Assertion, Deep Learning**

I. INTRODUCTION

Unit testing is a crucial activity of software development, which involves testing an individual unit of software applications, such as a method, a class, or a module. While integration and system testing assess the overall performance of a system, unit testing focuses on validating that each unit of code works as intended and conceived by the developer, thereby detecting and diagnosing failures before they propagate throughout the system and preventing regressions. Effective unit tests can improve the quality of software, reduce the incidence, and cost of software failures [1], [2], as well as enhance the entire software development process.

```
1 @Test
2 public void test01() throws Throwable  {
3     CategoryAxis3D categoryAxis3D0 = new CategoryAxis3D();
4     categoryAxis3D0.setVisible(false);
5     CategoryAxis3D categoryAxis3D1 = new CategoryAxis3D();
6     boolean boolean0 = categoryAxis3D1.equals(categoryAxis3D0);
7     assertFalse(categoryAxis3D0.isVisible());
8     assertFalse(boolean0);
9 }
```

Fig. 1. Example of a unit test case.

A unit test comprises a *test prefix*, which is a series of statements that manipulate a unit under test to attain a specific state, and a *test oracle*, which typically includes an assertion that specifies the expected behavior under that state [3]. For instance, in the unit test illustrated in Figure 1, lines 3-6 constitute the test prefix, which creates two `CategoryAxis3D` objects, with `categoryAxis3D0` set to invisible; testers utilize a boolean variable to check whether `categoryAxis3D0` and `categoryAxis3D1` are equal. On lines 7-8, the assertion specifies the expected outcome, which tests that after executing the test prefix, the visibility property of `categoryAxis3D0` should be `False` and that the two objects should not be equivalent.

Despite the significant benefits of testing, creating effective unit tests is a non-trivial and time-consuming task. Previous studies have indicated that developers can spend more than 15% of their time on test generation [4]. To streamline unit test generation, various automated testing tools have been proposed, such as Randoop [5] and EvoSuite [6]. However, these test-generation tools prioritize generating high-coverage tests over meaningful assertions and face challenges in comprehending the intended program behavior. For example, an industrial evaluation [7] of assertions generated by EvoSuite has shown that "in manually written tests, the assertions are meaningful and useful unlike the generated ones." As a result, a lot of manual effort is still required in conducting unit testing.

To reduce the effort for oracle generation, Watson et al. introduced ATLAS [8], a deep learning (DL)-based technique that trains a neural generative model on an extensive corpus of existing unit tests. ATLAS operates by taking a pair consisting of a test prefix and its *focal method* (i.e., the method

* Both authors contributed equally to this research
† Meng Yan is the corresponding author

under test) that encompasses both the method names and the method bodies. For consistency with prior research [9], we refer to such a pair as *focal-test*. ATLAS then generates an assertion for the focal-test from scratch. Neural models, unlike specification-mining-based approaches, are more flexible, particularly when documentation is imprecise or incomplete. Thus, ATLAS offers a more adaptable solution to the test assertion problem. However, the effectiveness of ATLAS is limited by several issues: 1) ATLAS generally prefers high-frequency words in the corpus and may have trouble with low-frequency words, such as project-specific identifiers. 2) ATLAS has poor performance when generating long sequences of tokens as assertions.

Recently, Yu et al. [9] proposed an information retrieval (IR)-based assertion generation method, including IR-based assertion retrieval (IR_{ar}) and retrieved-assertion adaptation (RA_{adapt}) techniques. IR_{ar} takes the same input as ATLAS and retrieves the most similar assertion to the given focal-test based on the Jaccard similarity coefficient [10]. Then, RA_{adapt} further adjusts the tokens in the retrieved assertion based on the context. Furthermore, an integrated approach [9] (abbreviated as *Integration*) that combines the IR-based approach with a DL-based approach has been proposed to improve assertion generation capabilities. The integrated method verifies the compatibility between the retrieved assertion and the current focal-test. If the compatibility exceeds a threshold, the retrieved assertion is returned as the final result. Otherwise, the DL-based method generates the assertion. Experimental results show that the integrated approach achieves higher accuracy and BLEU score than ATLAS. While the performance is a notable achievement in assertion generation research, knowledge gaps still exist regarding why or where the proposed technique is effective or ineffective.

To fill this gap in the literature, we first conduct a comprehensive evaluation of *Integration* to gain a better understanding of its application scenarios. We conduct the empirical assessment on two public datasets adopted by Yu et al. [9], namely $Data_{old}$ and $Data_{new}$. The main findings are:

- *Integration* relies mainly on the IR-based method for generating assertions. For instance, *Integration* utilizes the IR-based approach to generate 80.06% of the assertions for $Data_{new}$. Additionally, for $Data_{old}$ and $Data_{new}$, 83.38% and 92.47% of correct assertions generated by the IR-based approach are identical to the retrieved assertions.
- *Integration* only replaces tokens during its adaptation operation, making it challenging to generalize to complex assertion generation scenarios.
- *Integration* incorrectly modifies assertions frequently, even when the retrieved assertion matches the ground truth exactly. When one token needs to be modified to get the correct result, the average accuracy of *Integration* is just 20.14% and drops to only 1.91% for more than five tokens.
- *Integration* fails to comprehend the semantic differences between focal-tests, resulting in many cases where retrieved assertions require modifications, but it is returned directly as expected assertion.

Overall, our empirical study reaffirms previous findings that based on the similarity of focal-test, we can always find "almost correct" assertions that are very similar to the correct ones. However, we also identify several limitations that restrict the effectiveness and generalizability of *Integration*. On the one hand, the single token replacement operation struggles with complex assertion edit scenarios, resulting in low accuracy (1.9%) of *Integration* when modifying more than five tokens. On the other hand, *Integration*'s inability to comprehend semantic differences between input and similar focal-tests causes it to make incorrect editing actions or fail to edit retrieved assertions when necessary.

To alleviate the above-mentioned limitations and achieve higher levels of performance, this paper proposes EDITAS, a novel retrieve-and-edit approach for assertion generation. The effectiveness of IR implies that similar focal-tests' assertions can be reused. In other words, certain tokens in the expected assertion are also highly probable to appear in the retrieved assertion. The improvements by the specification-mining-based and retrieved-assertion adaptation approaches have revealed the significance of assertion patterns. Inspired by this, EDITAS views the assertion from a similar focal-test as a prototype and leverages a neural sequence-to-sequence model to learn the assertion edit patterns used to modify the prototype. Our motivation is that the retrieved assertion guides the neural model on "how to assert" and the assertion edit pattern highlights to the neural model "what to assert".

EDITAS consists of two major components: a Retrieval component and an Edit component. Given an input focal-test, the Retrieval component obtains its similar focal-test from a corpus and utilizes the retrieved focal-test's corresponding assertion as a prototype. In the Edit component, a sequence-to-sequence neural network is trained to edit the prototype, based on edit sequences representing the semantic differences between the input and the similar focal-test. On the one hand, EDITAS effectively mitigates the long assertion generation problem, which is the performance bottleneck of ATLAS, by editing assertions instead of generating them from scratch. Additionally, EDITAS is more generalizable than *Integration* because it can 1) comprehensively understand the semantic differences between focal-tests; 2) apply appropriate assertion edit patterns with greater flexibility; and 3) generate more diverse edit actions than just replacement operations.

The IR-based assertion retrieval method (IR_{ar}), the retrieved-assertion adaptation methods including RA^{H}_{adapt} and RA^{NN}_{adapt}, the integrated approach *Integration*, and ATLAS are used as baselines. We evaluate EDITAS and the baselines on both $Data_{old}$ and $Data_{new}$ datasets in terms of accuracy and BLEU. The evaluation results show that EDITAS outperforms all baselines across all metrics in assertion generation. Specifically, EDITAS achieves the highest accuracy, outperforming the baselines by 14.87%-70.15% and 5.12%-104.80% on the two datasets, respectively.

In summary, the contributions of this paper include:

(1) We conduct an in-depth analysis of the state-of-the-art assertion generation method that combines DL and IR techniques. Our analysis results provide valuable insights for future research in this direction.
(2) We propose a novel retrieve-and-edit approach, namely EDITAS, for assertion generation. EDITAS utilizes assertions from similar focal-tests as prototypes and uses a neural sequence-to-sequence model to learn the assertion edit patterns.
(3) We conduct extensive experiments to evaluate our approach. The experimental results show that EDITAS significantly outperforms all baselines.
(4) We open-source our replication package[1], including the dataset, the source code of EDITAS, the trained model, and assertion generation results, for follow-up studies.

II. Background and Related Work

A. DL-based Assertion Generation

With the rise of deep learning (DL), an increasing number of software engineering tasks can be effectively tackled using advanced DL techniques, such as code search [11], [12], automated program repair [13]–[17], fault diagnosis [18], code summarization [19]–[22], and code clone detection [23]–[27]. Watson et al. [8] recently proposed ATLAS, the first DL-based assertion generation method. ATLAS employs Neural Machine Translation (NMT) to generate assertions for a given *focal-test*, which consists of a test method without any assertion (i.e., test prefix) and its focal method (i.e., the method under test), including both method names and method bodies. To develop ATLAS, Watson et al. first extracted Test-Assert Pairs (TAPs) from GitHub projects that use the JUnit testing framework. Each pair consists of a focal-test and its corresponding assertion. The initial TAP set is then preprocessed into two datasets: 1) raw source code, where TAPs are simply tokenized (see the top part of Figure 2), and 2) abstract code, where uncommon tokens are further represented by their respective types and sequential IDs from the raw source code (see the bottom part of Figure 2). Ultimately, ATLAS generates a meaningful assertion to verify the correctness of the focal method.

Recent research [28]–[30] has explored the potential of pre-trained models, such as T5 and BART, for supporting the assertion generation task through pre-training and fine-tuning. Specifically, these approaches involve pre-training a transform model using a large corpus of either source code or English language, followed by fine-tuning it for assertion generation. Dinella [3] proposed TOGA to address the assertion generation problem by using a ranking architecture over a set of candidate test assertions. TOGA employs grammar along with type-based constraints to limit the generation space of candidates and uses a transformer-based neural approach to obtain the most probable assertions. We find that the formal implementation of TOGA [31], [32] requires a seed/approximate assertion to determine the variables for constructing and optimizing the candidate assertion set. Also, the seed/approximate assertion needs to be created manually or via EvoSuite tool [6]. Unlike TOGA, this paper focuses on the problem of assertion generation using only focal-test.

[1]https://github.com/swf-cqu/EditAS

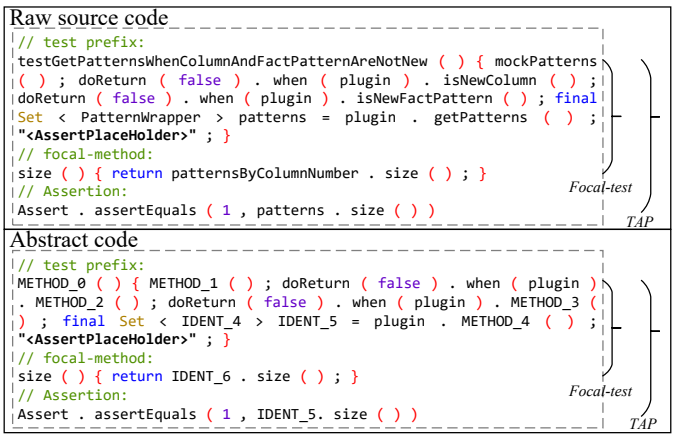

Fig. 2. Abstraction process.

B. Combining Information Retrieval and Deep Learning for Assertion Generation

The community has been working to boost DL techniques in software engineering tasks by leveraging Information Retrieval (IR) techniques and achieved promising results. Drawing on the idea of "combining IR and DL", Yu et al. [9] proposed a novel integration approach to tackle the assertion generation problem. Their contributions include two IR-based approaches for assertion generation, namely IR-based assertion retrieval (IR_{ar}) and retrieved-assertion adaptation (RA_{adapt}).

The basic idea of IR_{ar} is to retrieve the assertion whose corresponding focal-test has the highest similarity (e.g., Jaccard [10] similarity) with the given focal-test, and the retrieved assertion is returned as an expected assertion. As IR_{ar} does not always retrieve completely accurate assertions, RA_{adapt} has been proposed to automatically adapt retrieved assertions to the right forms utilizing contextual information. For a retrieved assertion, RA_{adapt} performs the following adaptation process: 1) deciding whether the assertion should be modified; 2) deciding which token (i.e., invoked method, variable, or constant) should be modified; 3) deciding what value a candidate token should be replaced with. Yu et al. have proposed two replacement strategies for determining the replacement value: one based on heuristics (RA^{H}_{adapt}) and the other on neural networks (RA^{NN}_{adapt}). RA^{H}_{adapt} utilizes lexical similarity for code replacement. In contrast to RA^{H}_{adapt}, RA^{NN}_{adapt} further enhances lexical similarity by incorporating semantic information using a neural network architecture and computes replacement values for code adaptation. Finally, Yu et al. [9] combine IR and DL techniques and propose an integrated approach, referred to as *Integration* in this paper. *Integration* first retrieves assertions using Jaccard similarity and adjusts the retrieved assertion if necessary. Then, *Integration* uses a semantic compatibility inference model to compute the "compatibility" of the adjusted assertion and the current

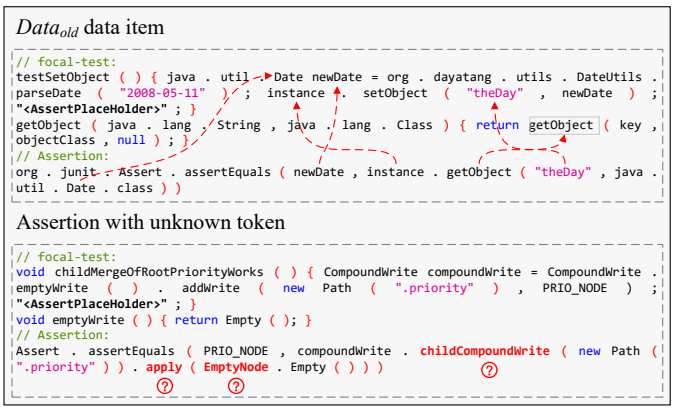

Fig. 3. Assertion with known *vs.* unknown tokens.

focal-test. If the compatibility is below a specified threshold (denoted as t), *Integration* switches to ATLAS to generate an assertion from scratch. The value of t is determined based on the validation set. Given that RA^{NN}_{adapt} outperforms other IR-based approaches, including IR_{ar} and RA^{H}_{adapt}, we adopt the combination of RA^{NN}_{adapt} and ATLAS to explore the optimal performance of *Integration*.

III. The Empirical Exploration of *Integration*

The empirical study aims to investigate the application scenarios of *Integration* [9] and gain insight into its mechanisms i.e., where and why *Integration* works and does not work. To this end, three research questions are designed.

- **RQ1**: *What are the characteristics of the dataset?* We first perform an in-depth analysis of the dataset. In particular, we analyze the distribution of assertion lengths on the entire dataset. Next, for each Test-Assertion Pair TAP_i in the test set, we retrieve the assertion from the training set whose corresponding focal-test has the highest similarity to the focus-test of TAP_i and compute the edit distance between the assertion of TAP_i and the retrieved assertion. Such investigation not only validates the dataset's characteristics but also helps us comprehend the intrinsic association between TAPs and their similar instances, which can guide our method design (see Section IV).
- **RQ2**: *Where and why does Integration work?* We further explore the advantages of *Integration* for assertion generation. We first classify the results generated by *Integration* according to the assertion generation method used. Next, we analyze the assertions that are generated correctly.
- **RQ3**: *Where and why does Integration fail?* We investigate the weaknesses of *Integration*, which are critical in that understanding the application scenario of an approach can better help us apply it in practice [33].

A. Dataset

We use the two publicly available datasets [34] from Yu et al. for our experiments, namely $Data_{old}$ and $Data_{new}$, respectively. To make our paper self-contained, we briefly describe the $Data_{old}$ and $Data_{new}$ in the following paragraphs.

1) $Data_{old}$: $Data_{old}$ is derived from the original dataset used by ATLAS. Initially, $Data_{old}$ is extracted from a pool of 2.5 million test methods in GitHub, which include test prefixes and their corresponding assertion statements. For each test method, $Data_{old}$ includes the focal method, i.e., the production code under test. The $Data_{old}$ is then preprocessed to exclude test methods with token lengths exceeding 1K and filter out assertions containing *unknown* tokens that are not present in the focal-test and the vocabulary, following established practice in natural language processing [35], [36]. As an example, the bottom part of Figure 3 highlights the unknown tokens `childCompoundWrite`, `apply`, and `EmptyNode`. After removing duplicates, $Data_{old}$ obtains 156,760 data items, which are further divided into training, validation, and test sets at an 8:1:1 ratio.

2) $Data_{new}$: The exclusion of assertions with unknown tokens can oversimplify the assertion generation problem, making $Data_{old}$ unsuitable for representing the realistic data distribution. This, in turn, poses a significant threat to the validity of experimental conclusions. Therefore, Yu et al. [9] constructed an expanded dataset, denoted as $Data_{new}$, by adding those cases that are excluded with unknown tokens in $Data_{old}$. Besides the existing data items in $Data_{old}$, $Data_{new}$ contains an extra 108,660 samples with unknown tokens to form a total of 265,420 data items, which are also divided into training, validation, and test sets in an 8:1:1 ratio.

B. Study Results

1) RQ1: What are the characteristics of the dataset: Figure 4 shows the distribution of assertion length, where the X axis represents various assertion lengths and the Y axis represents the number of corresponding assertions. Both datasets reveal a similar distribution trend: Assertions are typically less than 30 tokens, and the number of assertions decreases as the length of the assertions increases. The average

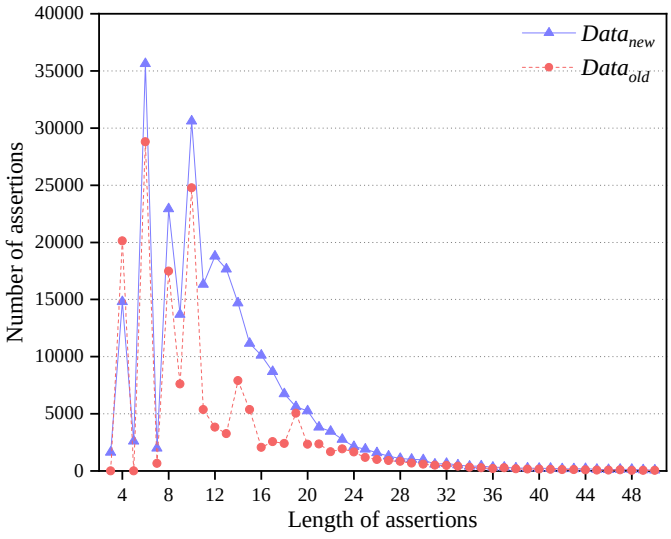

Fig. 4. Length distribution of assertions of each dataset.

length of assertions in $Data_{new}$ is 13 tokens, while in $Data_{old}$ that is 12 tokens. This finding suggests that in practice, testers may prefer brief assertions as long ones can cause maintenance difficulties.

Finding-1 of RQ1

Both $Data_{new}$ and $Data_{old}$ exhibit a long-tail distribution in assertion length and edit distance. The majority of assertion lengths are concentrated within a small number of tokens.

TABLE I
EDIT DISTANCE (E) BETWEEN RETRIEVED ASSERTIONS AND GROUND TRUTH FOR EACH TEST SET

Dataset	$E=0$	$E=1$	$E=2$	$E=3$	$3<E\leq 5$	$5<E\leq 10$	$10<E$	Total
$Data_{old}$	5,684	2,377	934	598	1,020	2,082	2,981	15,676
$Data_{new}$	10,059	3,059	1,581	1,031	1,563	3,050	6,199	26,542

Table I further provides edit distances between test samples' ground truth (i.e., expected assertions) with retrieved assertions. Interestingly, we can observe that a significant proportion of assertions match exactly with retrieved assertions, accounting for $37.90\% = 10,059/26,542$ and $36.26\% = 5,684/15,676$ in $Data_{new}$ and $Data_{old}$, respectively. In addition, for $Data_{new}$ and $Data_{old}$, 65.15% and 67.70% of the test samples only need to modify the retrieved assertion by less than or equal to five tokens to produce a correct assertion. Such findings provide evidence of the effectiveness of the information retrieval (IR) approach: by identifying similarities from the focal-tests, we can find almost or completely matched assertions.

Finding-2 of RQ1

The Information Retrieval (IR) technique can successfully search nearly or completely correct assertions by identifying the similarity of focal-tests.

2) RQ2: Where and why does Integration work: To gain insight into where and why *Integration* works, we analyze its prediction results. Given that *Integration* is an integrated method that integrates ATLAS and an IR-based assertion generation method (RA^{NN}_{adapt} in this paper), we categorize *Integration*'s generated results based on the assertion generation method used. Our analysis reveals that RA^{NN}_{adapt} generates 21,250 ($80.06\% = 21,250/26,542$) assertions for $Data_{new}$ and 10,215 ($65.16\% = 10,215/15,676$) assertions for $Data_{old}$. The results indicate that *Integration* prefers assertions generated by the IR-based approach.

Finding-1 of RQ2

The effectiveness of *Integration* is primarily dependent on the IR-based assertion generation method.

As the IR-based approach contributes most to the *Integration*, we conduct further analysis on the correct assertions generated by RA^{NN}_{adapt}. Table II shows the edit distance between retrieved assertions and correct ones generated by RA^{NN}_{adapt} for $Data_{new}$ and $Data_{old}$, accordingly. From the table, it can be seen that the success of RA^{NN}_{adapt} is mainly based on retrieving assertions that are identical to the ground truth, such as the samples with $E=0$ account for $92.47\% = 9,839/10,640$ in $Data_{new}$. RA^{NN}_{adapt} enhances the assertion generation of *Integration* through adaptation operations, but is more effective for single or two token modifications than for other modification cases. For example, with the $Data_{new}$ dataset, RA^{NN}_{adapt} achieves an accuracy of $15.21\% = 437/(437+2,436)$ when only one token is altered and $7.75\% = 67/(67+798)$ when three tokens are modified.

TABLE II
EDIT DISTANCE (E) BETWEEN RETRIEVED ASSERTIONS AND ASSERTIONS GENERATED BY RA^{NN}_{adapt}

Dataset	Prediction	$E=0$	$E=1$	$E=2$	$E=3$	$E=4$	$E=5$	$E>5$	Total
$Data_{old}$	*Correct*	5,435	486	341	108	73	35	40	6,518
	Incorrect	52	1,453	342	255	224	226	1,145	3,697
$Data_{new}$	*Correct*	9,839	437	235	67	32	8	22	10,640
	Incorrect	130	2,436	1,162	798	709	543	4,832	10,610

Finding-2 of RQ2

The majority of the successful assertions produced by RA^{NN}_{adapt} are exactly the same as the retrieved ones. In terms of adaptation operations, RA^{NN}_{adapt} performs better on single token modifications than other modification cases.

3) RQ3: Where and why does Integration fail: Previous work [9] has demonstrated that one of the bottlenecks of ATLAS's performance arises from generating assertions with long sequences. Hence, in RQ3, we focus on exploring where and why RA^{NN}_{adapt} fails. We first collect the incorrect assertions generated by RA^{NN}_{adapt}, and calculate the edit distance between retrieved assertions and their ground truth. As shown in Table II, for $Data_{new}$, there are 130 test samples whose assertions match the ground truth but are still modified by RA^{NN}_{adapt}, and a similar phenomenon occurs in $Data_{old}$. The performance of RA^{NN}_{adapt}'s adaptation strategy is limited. For instance, when $E=1$, RA^{NN}_{adapt} incorrectly generates $84.79\% = 2,436/(437+2,436)$ of assertions for $Data_{new}$ and $74.94\% = 1,453/(1,453+486)$ for $Data_{old}$. The performance of RA^{NN}_{adapt} is worse when $E>5$, with an average accuracy of only 1.91%.

Finding-1 of RQ3

Integration frequently incorrectly modifies assertions even when the retrieved assertion matches the ground truth exactly. When modifying one token to get the correct assertion, the average accuracy is only 20.14%.

We further count the number of edits made by RA^{NN}_{adapt} for those incorrect assertions. As reported in Table III, we can observe that there is a significant proportion of samples for which RA^{NN}_{adapt} does not make any changes to their retrieval assertions, e.g. in $Data_{new}$, such type of sample size is

TABLE III
NUMBER OF EDITS (N) MADE BY RA^{NN}_{adapt} FOR INCORRECT ASSERTIONS

Dataset	$N=0$	$N=1$	$N=2$	$N=3$	$N=4$	$N=5$	$N>5$	Total
$Data_{old}$	3,071	244	173	90	51	26	42	3,697
$Data_{new}$	9,436	497	328	167	73	35	74	10,610

9,436, representing 88.93% of the total number of incorrect assertions. Our analysis is that this may be due to the difficulty of RA^{NN}_{adapt} in understanding the semantic differences between focal-tests (an example is shown in Figure 7). Yu et al. [9] argue that an assertion needs to be edited if it contains at least one token absent from the input focal-test. However, this setting ignores many instances where the necessary changes are made to adapt the context of the input focal-test, even if all the tokens of the retrieved assertions occur in the input focal-test.

Finding-2 of RQ3

Integration fails to comprehend the semantic differences between focal-tests, resulting in many cases where retrieved assertions require modifications but are not edited appropriately.

IV. RETRIEVE-AND-EDIT ASSERTION GENERATION

Drawing on insights garnered from our empirical study, we propose EDITAS, a retrieve-and-edit approach for generating assertions. Unlike *Integration*, which only considers replacement operations to adjust retrieved assertions, EDITAS can learn diverse assertion edit patterns from similar TAPs. Given a focal-test as input, EDITAS retrieves a corpus to obtain the similar TAP based on the similarity of focal-test and generates a new assertion via the retrieved TAP's assertion and compatible edit patterns. The overall framework of EDITAS is illustrated in Figure 5. EDITAS consists of a Retrieval component and an Edit component.

A. Retrieval component

In our approach, the Retrieve component retrieves a similar Test-Assert Pair (TAP) from a corpus given the input focal-test. Specifically, to facilitate efficient retrieval, EDITAS first tokenizes each focal-test in the training and test sets using javalang [37] and removes duplicate tokens. Then, the Retrieve component retrieves the TAP whose focal-test has the highest Jaccard similarity coefficient with the input focal-test. The Jaccard similarity is a text similarity measurement that considers the overlap of words between two texts, calculated using the following formula:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

Where A and B are two bags of words, $|\cdot|$ denotes the number of elements in a collection.

B. Edit component

We employ a neural edit model to learn diverse assertion edit patterns from similar TAPs. The edit model is designed to understand how to modify one assertion to another based on semantic differences between focal-tests. Specifically, for a given focal-test ft and its similar focal-test instance ft', along with their corresponding assertions x and y, the neural edit model aims to find a function f such that $f\,(ft, ft', x) = y$. In this section, we elaborate on the focal-test semantic difference representation and the neural edit model training.

1) Focal-test semantic difference representation: We extract and compare semantic information and modification details between focal-tests using edit sequences, according to the previous work's finding [38]: different words between the two methods can reflect their semantic differences. We follow a similar approach as in [39], [40], aligning the two tokenized focal-test sequences using a diff tool and creating an edit sequence based on the resulting alignment. As shown in Figure 6, each element (named as an *edit*) in an edit sequence is represented as a triple $\langle ft_i, ft_i', a_i \rangle$, where ft_i is a token in one focal-test and ft_i' is a token in the similar focal-test, and a_i is the edit action that transforms ft_i to ft_i'. There are four types of edit actions: *insert*, *delete*, *equal*, or *replace*. when a_i is an *insert* (*delete*) operation, it means that ft_i (ft_i') will be an empty token $\emptyset$. Constructing such an edit sequence can not only preserve the information of the focal-test (i.e., ft_i and ft_i') but also highlight their fine-grained differences through a_i.

2) Neural edit model training: Our neural edit model is fundamentally a sequence-to-sequence neural architecture. It accepts a retrieved assertion $x = \left[x_1, x_2, \cdots, x_{|x|}\right]$ and an edit sequence $E = \left[\left\langle ft_1, ft_1^{'}, a_1 \right\rangle, \ldots \left\langle ft_n, ft_n^{'}, a_n \right\rangle\right]$ as input and is designed to generate a new assertion $y = \left[y_1, y_2, \cdots, y_{|y|}\right]$. Specifically, EDITAS incorporates two encoders: the Edit Sequence Encoder and the Assertion Encoder, to respectively encode the edit sequence and the retrieved assertion. An attention mechanism is then applied on the encoder side to learn the relationship between the edit sequence and the retrieved assertion. The decoder consists of two pointer generators that enable the model to copy tokens from both the input focal-test and the retrieved assertion concurrently during the generation process. This approach can effectively preserve the original meaning of the retrieved assertion while integrating the assertion edit patterns reflected in the edit sequence.

❶ **Encoders**. The structures of the two encoders, i.e., the Edit Sequence Encoder and the Assertion Encoder, are nearly identical. They consist of a contextual embedding layer, an attention layer, and a modeling layer.

The Contextual Embed Layer. We first map the focal-test tokens, assertion tokens, and edit action tokens to embeddings. Considering there are only four edit actions, we randomly initialize an embedding matrix and update it during training. To capture both syntactic and semantic information, we employ a pre-trained model, such as fastText [41], to acquire word

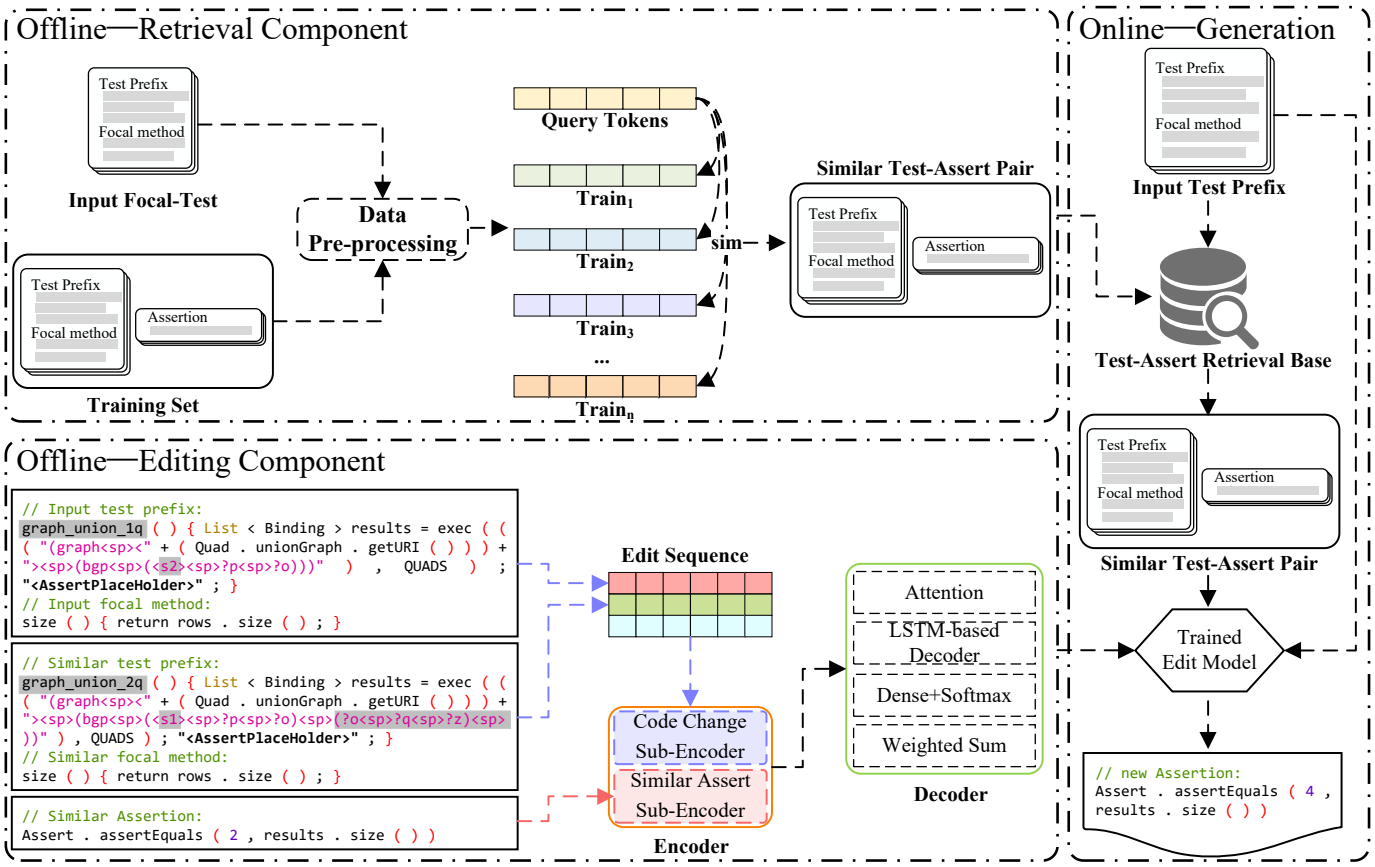

Fig. 5. The overall framework of our approach.

embeddings for each focal-test and assertion token. We then use a bidirectional long short-term memory (Bi-LSTM) [42] to process the sequence of word embeddings to access contextual information. For Edit Sequence Encoder and each edit E_i, the three embeddings, i.e., e_{ft_i}, $e_{ft_i'}$, e_{a_i} are first concatenated horizontally, and then input to the Bi-LSTM, as follows:

$$e_i^{'} = [e_{ft_i} \oplus e_{ft_i'} \oplus e_{a_i}]$$
$$\overrightarrow{h}_i^{'} = \text{LSTM}(\overrightarrow{h}_{i-1}^{'}, e_i^{'}); \overleftarrow{h}_i^{'} = \text{LSTM}(\overleftarrow{h}_{i+1}^{'}, e_i^{'})$$
$$h_i^{'} = [\overrightarrow{h}_i^{'} \oplus \overleftarrow{h}_i^{'}]$$

where $h_i^{'}$ is the contextual vector of this edit and $\oplus$ is a concatenation operation. Similarly, Assertion Encoder obtains the contextual vector h_i of each assertion token x_i with x_i's embedding e_{x_i} as input.

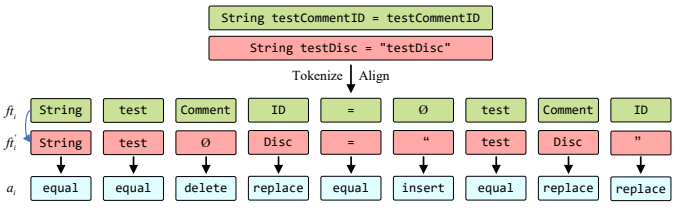

Fig. 6. Converting a difference between focal-tests to an edit sequence.

The Attention Layer. This layer is responsible for linking and fusing the information of the focal-test difference and that of the retrieved assertion, capturing their relationship. We place it on the top of the two contextual embed layers. The attention layer takes as input the contextual vectors, i.e., H and H', and outputs an assertion-aware (edit-aware) feature vector for each edit (assertion token), as well as the original contextual vector for this edit (assertion token). Formally, the edit-aware feature vector g_i of assertion token x_i is calculated using the dot-production attention mechanism [43] as follows:

$$g_i = H^{'}\alpha_i; \quad \alpha_i = \text{softmax}(H^{'\top} W_\alpha h_i) \tag{2}$$

The attention weight α_i measures the importance of each edit is with respect to x_i. The computation of the assertion-aware feature vector $g_i^{'}$ of edit E_i is almost same and can be expressed as follows:

$$g_i^{'} = H\alpha_i^{'}; \quad \alpha_i^{'} = \text{softmax}(H^{\top} W_\alpha^{\top} h_i^{'})$$

The Modeling Layer. This layer uses two distinct Bi-LSTM to generate the final feature representation based on the contextual vector of each edit (assertion token) and the assertion-aware (edit-aware) feature vector, respectively. For

example, given an assertion token x_i, its final representation z_i is computed as follows:

$$f_i = [g_i \oplus h_i]$$
$$\overrightarrow{z}_i = \text{LSTM}(\overrightarrow{z}_{i-1}, f_i); \overleftarrow{z}_i = \text{LSTM}(\overleftarrow{z}_{i+1}, f_i)$$
$$z_i = [\overrightarrow{z}_i \oplus \overleftarrow{z}_i]$$

❷ **Decoder**. EDITAS uses an LSTM-based decoder to generate a new assertion from the outputs of two encoders, Z and $Z^{'}$. The final hidden states of both modeling layers are concatenated and used as the initial state for the decoder's LSTM. During decoding step j, the decoder computes the hidden state s_j based on the j-th word embedding of ground truth assertion $e_{\hat{y}_j}$, the previous hidden state s_{j-1}, and the previous output vector o_{j-1}, as follows:

$$s_j = \text{LSTM}(s_{j-1}, [e_{\hat{y}_j} \oplus o_{j-1}]) \quad (3)$$

We compute a context vector at each time step as the representation of the encoder's input by the dot-product attention mechanism, following Equation 2. Given there are two encoders, the decoder obtains two context vectors, i.e., c_j from the retrieved assertion and $c_j^{'}$ from the focal-test difference. An output vector o_j is computed using c_j, $c_j^{'}$ and s_j, and the corresponding vocabulary distribution P_j^{vocba} is obtained using a softmax layer.

$$o_j = tanh(V_c[c_j \oplus c_j^{'} \oplus s_j]; \quad P_j^{vocba} = \text{softmax}(V_c^{'} o_j)$$

where V_c and $V_c^{'}$ are trainable parameters. P_j^{vocba} records the probability of each token being generated, of which the element with the highest probability will be the output under decoding step j. Due to the similarity of focal-tests, it is reasonable to assume that certain tokens in the new assertion should also appear in the retrieved assertion, while others that are not present in the retrieved assertion should be included in the input focal-test. Therefore, we adopt the pointer generator to copy tokens from the retrieved assertion and the input focal-test:

$$P_j^{ass}(y_j) = \sum_{k:x_k=y_j} \beta_{jk}; \quad P_j^{ft}(y_j) = \sum_{k:t_k^{'}=y_j} \beta_{jk}^{'}$$

$P_j^{ass}(y_j)$ and $P_j^{ft}(y_j)$ are the probabilities of copying y_j from the retrieved assertion and the input focal-test, respectively. β_{jk} and $\beta_{jk}^{'}$ are the attention weights of x_k and E_k at time step j, computed based on the context vectors c_j and $c_j^{'}$.

The conditional probability of y_j at time step j is then the combination of P_j^{vocba}, P_j^{ass}, and P_j^{ft}, i.e.,

$$p(y_j|y_{<j}, x, E) = \gamma_j P_j^{vocba}(y_j) + (1-\gamma_j)(\theta_j P_j^{ass}(y_j) + (1-\theta_j) P_j^{ft}(y_j))$$

γ_j and θ_j represent the probabilities of generating y_j by selecting from the vocabulary and copying from the retrieved assertion, respectively. Both probabilities are modeled by a single-layer feed-forward neural network, which is trained jointly with the decoder.

C. Generation

Given an input focal-test ft, EDITAS generates an assertion through three steps:

Step 1: Selecting a similar TAP. EDITAS leverages a large-scale training dataset as the retrieval corpus. Then, the Retrieval component retrieves the TAP whose focal-test closely matches ft from the corpus. Further information on the retrieval process is outlined in Section IV-A.

Step 2: Capturing fine-grained semantic differences between focal-tests. Through Step 1, EDITAS obtains a similar focal-test to ft and its corresponding assertion statements. EDITAS calculates the edit sequences to capture semantic differences between ft and the retrieved focal-test. The details of this part are described in Section IV-B.

Step 3: Combining retrieved assertion with semantic differences between focal-tests. Finally, the trained neural edit model adjusts the retrieved assertion based on the edit sequences, thereby generating one assertion corresponding to ft. Further details on this model can be found in Section IV-B.

V. EXPERIMENTAL SETUP

In this section, we describe the dataset and evaluation metrics used in the experiment. We conduct experiments to answer the following research questions:

- **RQ4:** How does the proposed EDITAS perform compared to state-of-the-art assertion generation baselines?
- **RQ5:** Do different similarity coefficients affect the performance of EDITAS?

A. Baselines

To answer the above-mentioned research questions, we compare EDITAS to five baselines. We first chose ATLAS, the first and classic neural network-based method for assertion generation. ATLAS utilizes a sequence-to-sequence model to generate assertions from scratch. Given that EDITAS aims to revisit and improve retrieval-augmented deep assertion generation methods, we further adopt the three state-of-the-art (SOTA) retrieval-based methods, including IR_{ar}, RA_{adapt}^{H}, and RA_{adapt}^{NN}. Finally, we provide the performance comparison between EDITAS and *integration* which is a SOTA retrieval-augmented deep assertion generation method. For more details please refer to Section II.

B. Datasets

We utilize $Data_{old}$ and $Data_{new}$ to evaluate the effectiveness of EDITAS and baseline methods following Yu et al [9]. Compared to $Data_{old}$, $Data_{new}$ adds the excluded cases with unknown tokens back to $Data_{old}$. For more details, please refer to Section III-A. Table IV provides detailed statistics of the test sets for the two datasets, including their distribution across different assertion types.

TABLE IV
DETAILED STATISTICS OF EACH TYPE IN $Data_{new}$ AND $Data_{old}$

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%)	1,071 (4%)	735 (3%)	362 (1%)	319 (1%)	3,030 (11%)

TABLE V
COMPARISONS OF OUR APPROACH WITH EACH BASELINE

Approach	$Data_{old}$		$Data_{new}$	
	Accuracy	**BLEU**	**Accuracy**	**BLEU**
ATLAS	31.42 (↑ 70.15%)	68.51 (↑ 17.90%)	21.66 (↑ 104.80%)	37.91 (↑ 67.40%)
IR_{ar}	36.26 (↑ 47.43%)	71.48 (↑ 13.00%)	37.90 (↑ 17.04%)	57.98 (↑ 9.45%)
RA^{H}_{adapt}	40.97 (↑ 30.49%)	73.28 (↑ 10.22%)	39.65 (↑ 11.88%)	59.81 (↑ 6.10%)
RA^{NN}_{adapt}	43.63 (↑ 22.53%)	73.95 (↑ 9.22%)	40.53 (↑ 9.45%)	59.81 (↑ 6.10%)
Integration	46.54 (↑ 14.87%)	78.86 (↑ 2.42%)	42.20 (↑ 5.12%)	60.92 (↑ 4.17%)
EDITAS	**53.46**	**80.77**	**44.36**	**63.46**

↑ denotes performance improvement of EDITAS against state-of-the-art baselines

TABLE VI
DETAILED STATISTICS OF OUR APPROACH AND EACH BASELINE FOR EACH ASSERT TYPE

Dataset	Approach	Total	AssertType								
			Equals	**True**	**That**	**NotNull**	**False**	**Null**	**ArrayEquals**	**Same**	**Other**
$Data_{old}$	ATLAS	4,925 (31%)	2,501 (32%)	966 (35%)	248 (17%)	598 (51%)	229 (23%)	236 (30%)	100 (33%)	47 (15%)	0 (0%)
	IR_{ar}	5,684 (36%)	2,957 (38%)	1,039 (37%)	449 (31%)	439 (38%)	314 (31%)	285 (36%)	111 (36%)	89 (29%)	**1 (50%)**
	RA^{H}_{adapt}	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}	6,839 (44%)	3,509 (45%)	1,225 (44%)	**551 (38%)**	610 (52%)	342 (34%)	341 (43%)	134 (44%)	126 (41%)	**1 (50%)**
	Integration	7,295 (47%)	3,714 (47%)	1,333 (48%)	546 (38%)	724 (62%)	348 (35%)	352 (44%)	148 (48%)	**129 (41%)**	**1 (50%)**
	EDITAS	**8,380 (53%)**	**4,131 (53%)**	**1,581 (57%)**	526 (36%)	**807 (69%)**	**577 (57%)**	**469 (59%)**	**167 (54%)**	122 (39%)	0 (0%)
$Data_{new}$	ATLAS	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}	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%)
	RA^{H}_{adapt}	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}	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%)
	Integration	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%)**
	EDITAS	**11,773 (44%)**	**5,339 (42%)**	**1,702 (47%)**	**1,304 (37%)**	**800 (62%)**	**523 (49%)**	**376 (51%)**	**172 (47%)**	**139 (44%)**	1,418 (47%)

C. Metrics

Consistent with prior work [8], [9], the following metrics are utilized in our experiment.

1) Accuracy: We use accuracy to evaluate the effectiveness of assertion generation techniques. Specifically, a generated assertion is considered accurate if and only if it exactly matches the ground truth. Accuracy determines the percentage of samples in which the generated output matches the expected output in terms of syntax.

2) BLEU: In line with previous studies [8], [9], we use the muti-BLEU to measure the similarity between the generated assertion and the ground truth. BLEU calculates the modified n-gram precision of a candidate sequence (i.e., the generated assertion) to the reference sequence (i.e., the ground truth), where n ranges from 1 to 4. The modified n-gram precision values are then averaged, and a penalty is applied for overly short sentences.

D. Implementation Details

In EDITAS, the hyper-parameter settings are determined based on the performance on our validation set. For the focal-test tokens, assertion tokens, and edit actions, we use 300-dimensional word embeddings. These embeddings are obtained from a fastText model pre-trained on Common Crawl and Wikipedia [44]. During training, the word embeddings are frozen. The hidden states of the Bi-LSTMs and the LSTM in our model have dimensions of 256 and 512, respectively. The Edit Sequence Encoder, Assertion Encoder, and Decoder are jointly trained to minimize the cross-entropy loss. We use the Adam optimizer [45] with a learning rate of 0.001 and clip the gradients norm by 5. The training is done with a batch size of 8 and a dropout [46] rate of 0.2 for all LSTM layers. We truncate the overlong input, where the length of the edit sequence and the focal-test is set to 512. We stop training after 5 trials, and the model with the best (smallest) validation perplexity is selected for evaluation. All of our approaches are built based on PyTorch framework [47]. We conduct all the experiments on a Ubuntu 20.04 with four NVIDIA GeForce RTX 3090 GPUs, one 32-core processor, and 256GB memory.

VI. EXPERIMENTAL RESULTS

A. RQ4: EDITAS *vs. Baselines*

***Overall effectiveness of* EDITAS**. We calculate the accuracy and BLEU scores between the assertions generated by different approaches and human-written assertions. The experimental results are presented in Table V. We notice that

ATLAS performs the worst among all approaches. This is mainly attributed to two reasons: 1) ATLAS, as a typical sequence-to-sequence DL model, suffers from exposure bias and gradient disappearance, leading to poor effectiveness in generating a long sequence of tokens as an assertion. As demonstrated in the previous work [9], ATLAS generates less than 15 tokens with an accuracy of 46.3%, and only 17.9% accuracy of generating tokens with more than 15 tokens. 2) ATLAS has a weaker ability to generate statements that contain unknown tokens, which significantly degrades its overall performance. IR_{ar} retrieves assertions from the corpus and uses them as output results, achieving better performance than ATLAS. This indicates that the similar focal-test's assertion contains some valuable and reusable information, which also demonstrates that it is reasonable for us to use the assertion of a similar focal-test as the prototype. RA^{H}_{adapt} and RA^{NN}_{adapt} further adjust the retrieved assertions to enhance the capability of the IR-based approach in generating assertions. However, as shown in Table V, the performance of adaptation operations of both RA^{H}_{adapt} and RA^{NN}_{adapt} is limited, especially for complex datasets. For instance, RA^{NN}_{adapt} can improve 20.33% of accuracy compared to IR_{ar}, while for $Data_{new}$, the improvement is only 6.94%. A similar observation holds for RA^{H}_{adapt}. *Integration* combines IR and DL techniques and achieves better accuracy and BLEU scores than ATLAS and IR-based assertion generation methods.

From Table V, EDITAS achieves a significant performance improvement over ATLAS, with an average accuracy improvement of 87.48% and a BLEU score improvement of 42.65% on both datasets. This is attributed to EDITAS using the rich semantic information from the retrieved assertions, rather than generating assertions from scratch. Our approach EDITAS outperforms IR-based baseline methods and *Integration* across all evaluation metrics. Specifically, compared to IR_{ar}, RA^{H}_{adapt}, RA^{NN}_{adapt}, and *Integration*, EDITAS achieves an average accuracy improvement of 32.24%, 21.19%, 15.99%, and 10.00%, respectively, which demonstrates the effectiveness of our edit module. As compared with IR-based baselines, EDITAS adopts the retrieved assertions as prototypes, and makes modifications by considering the semantic difference between input and similar focal-tests. By combining the advantages of neural networks and IR-based methods, EDITAS achieves the best performance.

Effectiveness on different assertion types. We further compare the effectiveness of EDITAS and baseline methods for different types of assertions. Each column in Table VI represents an assertion type, and each cell shows the number of correctly generated assertions and their corresponding ratios in brackets. The results present that EDITAS performs better than the baseline methods for almost all assertion types, especially for the standard JUnit assertion type. For the other assertion type, EDITAS performs marginally worse than *Integration*. This may be attributed to the fact that the training set has a large of samples of the JUnit testing framework, leading EDITAS to learn more of the syntactic features of JUnit. Overall, the experimental results can indicate the generality of EDITAS in generating different types of assertions.

Input Focal-Test

```
// test prefix:
testToURLs3a ( ) { final  File [ ]
files = new File [ 0 ] ; final URL
[ ] urls = FileUtils . toURLs
( files ) ; "<AssertPlaceHolder>" ; }

// focal-method:
toURLs ( File [ ] ) { final URL [ ]
urls = new URL [ files . length ] ;
for ( int i = 0 ; i < ( urls .
length ) ; i ++ ) { urls [ i ] =
files [ i ] . toURI ( ) . toURL
( ) ; } return urls ; }
```

Retrieved Focal-Test

```
// test prefix:
testToFiles3 ( ) { final URL [ ] urls
= null ; final File [ ] files =
FileUtils . toFiles ( urls ) ;
"<AssertPlaceHolder>" ; }
// focal-method:
toFiles ( URL [ ] ) { if ( ( urls ==
null ) || ( ( urls . length ) ==
0 ) ) { return FileUtils .
EMPTY_FILE_ARRAY ; } final File [ ]
files = new File [ urls . length ] ;
for ( int i = 0 ; i < ( urls .
length ) ; i ++ ) { final URL url =
urls [ i ] ; if ( url != null ) { if
( ( url . getProtocol ( ) . equals
( "file" ) ) == false ) { throw new
IllegalArgumentException ( ( "URL
could not be converted to a File: " +
url ) ) ; } files [ i ] = FileUtils .
toFile ( url ) ; } } return files ; }
```

Retrieved Assertion: `org . junit . Assert . assertEquals ( 0 , files . length )`

IR_{ar}: `org . junit . Assert . assertEquals ( 0 , files . length )`

RA^{H}_{adapt}: `org . junit . Assert . assertEquals ( 0 , files . length )`

RA^{NN}_{adapt}: `org . junit . Assert . assertEquals ( 0 , files . length )`

Integration: `org . junit . Assert . assertEquals ( 0 , files . length )`

EDITAS: `org.junit.Assert.assertEquals ( 0 , urls . Length )`

Ground Truth: `org.junit.Assert.assertEquals ( 0 , urls . Length )`

Fig. 7. An example of generated assertions by our approach and baselines.

To better understand why EDITAS outperforms RA^{H}_{adapt} and RA^{NN}_{adapt}, we manually inspect the assertion generation results. Our analysis reveals that EDITAS has the following advantages over the other methods: 1) EDITAS is capable of learning and applying diverse assertion editing patterns, whereas RA^{H}_{adapt} and RA^{NN}_{adapt} cannot handle token addition or deletion operations. 2) RA^{H}_{adapt} and RA^{NN}_{adapt} only modify the retrieved assertion when it contains at least one token not present in the input focal-test. However, even if all tokens in the retrieved assertion appear in the input focal-test, it may still require modification due to the semantic differences between the focal-tests. In contrast, EDITAS leverages a probabilistic model to learn common patterns of assertion edits from existing focal-tests' semantic differences. Overall, the edit patterns learned by EDITAS are more diverse and can cover a wider range of samples. For example, Figure 7 presents a test sample. In this sample, the focal method of the input focal-test is responsible for transforming the file array into a URL array. On the other hand, the retrieved focal-test performs the opposite operation. Therefore, the corresponding test prefix constructs an empty URL array to verify that the transformed files array should also be empty. Similarly, the test prefix of the input focal-test constructs an empty files array, and its corresponding assertion aims to verify that the transformed URL array is empty. Methods including IR_{ar}, RA^{H}_{adapt}, RA^{NN}_{adapt}, and *Integration* encounter difficulty in understanding the semantic difference between focal-tests, as they fail to recognize the opposite operation being performed. In addition, the tokens for the retrieved assertions all appear in the input focal-test, which does not satisfy the condition that such methods modify the assertions. Consequently, they directly take the retrieved assertion as the expected result, leading to potential inaccuracies, while EDITAS succeeds.

Summary of RQ4

EDITAS significantly outperforms all baseline methods in accuracy and BLEU, with average performance improvement of 10.00%-87.48% and 3.30%-42.65% on the two datasets, respectively.

B. RQ5: The effectiveness with different similarity coefficients

EDITAS utilizes Jaccard as its default similarity coefficient. In this RQ, we aim to examine whether the similarity coefficients affect EDITAS's effectiveness. Specifically, we developed two additional versions of EDITAS, utilizing two commonly used similarity coefficients, Dice [48] and Overlap [49], respectively. Given two sets X and Y, Dice and Overlap calculate the similarity as follows.

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

Table VII displays the accuracy of EDITAS on both datasets using different similarity coefficients. Our results illustrate that the similarity coefficient does have an impact on the effectiveness of EDITAS. Specifically, Jaccard and DICE similarity coefficients have little impact on the effectiveness of EDITAS. Notably, we observe that Overlap yields the poorest accuracy and BLEU scores compared to Jaccard and DICE. We attribute this to Overlap only accounting for the degree of overlap between the two focal-tests, ignoring their differences.

TABLE VII
PERFORMANCE OF DIFFERENT SIMILARITY COEFFICIENTS

Dataset	Metrics	Jaccard	Overlap	DICE
$Data_{old}$	Accuracy	53.46	46.13	53.46
	BLEU	80.77	75.42	80.77
$Data_{new}$	Accuracy	44.36	38.22	44.24
	BLEU	63.46	56.37	63.59

Summary of RQ5

The similarity coefficient does have an impact on the effectiveness of EDITAS. Improving the Retrieval component's capability can lead to better assertion generation performance of EDITAS.

VII. THREATS TO VALIDITY

There are three main threats to the validity of our approach. Firstly, following previous works [8], [9], we only conducted experiments on two Java datasets. Although Java may not be representative of all programming languages, the datasets used in our experiments are large enough and provide sufficient safety to demonstrate the effectiveness of EDITAS. Furthermore, EDITAS employs exclusively language-agnostic features, i.e., edit sequences, and can be applied to other programming languages. Secondly, the Retrieve component retrieves a similar focal-test based on lexical similarity. This may result in the retrieved focal-test and input focal-test being similar only at the lexical level while exhibiting different assertions. To address this threat, we use a large-scale Java dataset (247M) to increase the scale and diversity of the retrieval corpus. We also propose an Edit component, which modifies the prototype by considering the semantic differences between the input focal-test and the retrieved focal-test, to alleviate this threat. The final major threat comes from the lack of comparing EDITAS with other existing assertion generation methods in terms of generating compilable assertions. While compilability can be used as another indicator of assertion quality, automated construction has been a challenging task that is highly dependent on external/internal settings/resources [50], [51]. Hence, automatically compiling large-scale assertions remains an obstacle. To ensure scalability, we do not use compilability and bug detection as metrics.

VIII. CONCLUSION AND FUTURE WORKS

In this paper, we reaffirm the previous finding that Information Retrieval (IR) can always find a "almost correct" assertion that is very similar to the expected one, and emphasize the shortcomings of previous approaches in modifying retrieved assertions. To alleviate these problems, we propose a novel retrieve-and-edit approach named EDITAS for assertion generation. EDITAS contains two components. A Retrieve component for retrieving the similar focal-test that consists of a test method without any assertion and its focal method (i.e., the method under test). An Edit component treats the assertion of a similar focal-test as a prototype and combines the prototype and assertion edit pattern reflected by semantic differences between the input focal-test and similar focal-test to generate a target assertion. We conducted extensive experiments on two large-scale Java datasets. The experimental results show that EDITAS substantially outperforms the state-of-the-art baselines. Our work shows that for the assertion generation task, retrieving similar assertions and learning to modify retrieved assertions by applying a set of editing operations can achieve satisfactory performance. In the future, we plan to explore additional techniques to enhance EDITAS further. One avenue we will explore is integrating contextual information or leveraging code language models to handle code changes more effectively. Additionally, we aim to extend our approach to support other programming languages, such as Python, to enhance the generalizability and applicability of our method. Our source code and experimental data are available at https://github.com/swf-cqu/EditAS.

IX. ACKNOWLEDGEMENTS

This work was supported in part by the National Key Research and Development Project (No. 2021YFB1714200), the Fundamental Research Funds for the CentralUniversities (No. 2022CDJDX-005), the Chongqing Technology Innovation and Application Development Project (No. CSTB2022TIAD-STX0007 and No. CSTB2022TIAD-KPX0067), the National Natural Science Foundation of China (No. 62002034) and the Natural Science Foundation of Chongqing (No. cstc2021jcyj-msxmX0538).

REFERENCES

[1] A. Hartman, "Is ISSTA research relevant to industry?" in *Proceedings of the International Symposium on Software Testing and Analysis, ISSTA 2002, Roma, Italy, July 22-24, 2002*. ACM, 2002, pp. 205–206. [Online]. Available: https://doi.org/10.1145/566172.566207

[2] S. Planning, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology*, vol. 1, 2002.

[3] E. Dinella, G. Ryan, T. Mytkowicz, and S. K. Lahiri, "TOGA: A neural method for test oracle generation," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 2130–2141. [Online]. Available: https://doi.org/10.1145/3510003.3510141

[4] E. Daka and G. Fraser, "A survey on unit testing practices and problems," in *25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014, Naples, Italy, November 3-6, 2014*. IEEE Computer Society, 2014, pp. 201–211. [Online]. Available: https://doi.org/10.1109/ISSRE.2014.11

[5] C. Pacheco and M. D. Ernst, "Randoop: feedback-directed random testing for java," in *Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada*. ACM, 2007, pp. 815–816. [Online]. Available: https://doi.org/10.1145/1297846.1297902

[6] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*. ACM, 2011, pp. 416–419. [Online]. Available: https://doi.org/10.1145/2025113.2025179

[7] M. M. Almasi, H. Hemmati, G. Fraser, A. Arcuri, and J. Benefelds, "An industrial evaluation of unit test generation: Finding real faults in a financial application," in *39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, Buenos Aires, Argentina, May 20-28, 2017*. IEEE Computer Society, 2017, pp. 263–272. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2017.27

[8] 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*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1398–1409. [Online]. Available: https://doi.org/10.1145/3377811.3380429

[9] 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," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 163–174. [Online]. Available: https://doi.org/10.1145/3510003.3510149

[10] T. Tanimoto, *An Elementary Mathematical Theory of Classification and Prediction*. International Business Machines Corporation, 1958. [Online]. Available: https://books.google.com.hk/books?id=yp34HAAACAAJ

[11] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep api learning," in *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, 2016, pp. 631–642.

[12] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 933–944.

[13] Z. Chen, S. Kommrusch, M. Tufano, L. Pouchet, D. Poshyvanyk, and M. Monperrus, "Sequencer: Sequence-to-sequence learning for end-to-end program repair," *IEEE Trans. Software Eng.*, vol. 47, no. 9, pp. 1943–1959, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2940179

[14] H. Hata, E. Shihab, and G. Neubig, "Learning to generate corrective patches using neural machine translation," *CoRR*, vol. abs/1812.07170, 2018. [Online]. Available: http://arxiv.org/abs/1812.07170

[15] A. Mesbah, A. Rice, E. Johnston, N. Glorioso, and E. Aftandilian, "Deepdelta: learning to repair compilation errors," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 2019, pp. 925–936. [Online]. Available: https://doi.org/10.1145/3338906.3340455

[16] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, "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 2018, Montpellier, France, September 3-7, 2018*. ACM, 2018, pp. 832–837. [Online]. Available: https://doi.org/10.1145/3238147.3240732

[17] Q. Zhang, C. Fang, Y. Ma, W. Sun, and Z. Chen, "A survey of learning-based automated program repair," *CoRR*, vol. abs/2301.03270, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2301.03270

[18] Y. Lou, Q. Zhu, J. Dong, X. Li, Z. Sun, D. Hao, L. Zhang, and L. Zhang, "Boosting coverage-based fault localization via graph-based representation learning," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 664–676. [Online]. Available: https://doi.org/10.1145/3468264.3468580

[19] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*. ACM, 2018, pp. 200–210. [Online]. Available: https://doi.org/10.1145/3196321.3196334

[20] B. Li, M. Yan, X. Xia, X. Hu, G. Li, and D. Lo, "Deepcommenter: a deep code comment generation tool with hybrid lexical and syntactical information," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. ACM, 2020, pp. 1571–1575. [Online]. Available: https://doi.org/10.1145/3368089.3417926

[21] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, "Summarizing source code with transferred API knowledge," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. ijcai.org, 2018, pp. 2269–2275. [Online]. Available: https://doi.org/10.24963/ijcai.2018/314

[22] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, "Retrieval-based neural source code summarization," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1385–1397.

[23] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*. IEEE, 2020, pp. 261–271. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054857

[24] H. Wei and M. Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. ijcai.org, 2017, pp. 3034–3040. [Online]. Available: https://doi.org/10.24963/ijcai.2017/423

[25] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," in *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019*, Y. Guéhéneuc, F. Khomh, and F. Sarro, Eds. IEEE / ACM, 2019, pp. 70–80. [Online]. Available: https://doi.org/10.1109/ICPC.2019.00021

[26] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 2019, pp. 783–794. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00086

[27] G. Zhao and J. Huang, "Deepsim: deep learning code functional similarity," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 2018, pp. 141–151. [Online]. Available: https://doi.org/10.1145/3236024.3236068

[28] M. Tufano, D. Drain, A. Svyatkovskiy, and N. Sundaresan, "Generating accurate assert statements for unit test cases using pretrained transformers," in *IEEE/ACM International Conference on Automation of Software Test, AST@ICSE 2022, Pittsburgh, PA, USA, May 21-22, 2022*. ACM/IEEE, 2022, pp. 54–64. [Online]. Available: https://doi.org/10.1145/3524481.3527220

[29] A. Mastropaolo, S. Scalabrino, N. Cooper, D. Nader-Palacio, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Studying the usage of text-to-text transfer transformer to support code-related tasks," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 336–347. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00041

[30] A. Mastropaolo, N. Cooper, D. Nader-Palacio, S. Scalabrino, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Using transfer learning for code-related tasks," *IEEE Trans. Software Eng.*, vol. 49, no. 4, pp. 1580–1598, 2023. [Online]. Available: https://doi.org/10.1109/TSE.2022.3183297

[31] "ToGA Artifact." https://github.com/microsoft/toga, 2022.

[32] "A issue bug of ToGA Artifact." https://github.com/microsoft/toga/issues/3, 2022.

[33] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. ACM, 2018, pp. 373–384. [Online]. Available: https://doi.org/10.1145/3238147.3238190

[34] "*Integration* Artifact." https://github.com/yh1105/Artifact-of-Assertion-ICSE22, 2022.

[35] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=SyyGPP0TZ

[36] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[37] C. Thunes, "Javalang." https://github.com/c2nes/javalang, 2019.

[38] J. Li, Y. Li, G. Li, X. Hu, X. Xia, and Z. Jin, "Editsum: A retrieve-and-edit framework for source code summarization," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 155–166. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678724

[39] Z. Liu, X. Xia, M. Yan, and S. Li, "Automating just-in-time comment updating," in *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 2020, pp. 585–597. [Online]. Available: https://doi.org/10.1145/3324884.3416581

[40] Z. Liu, X. Xia, D. Lo, M. Yan, and S. Li, "Just-in-time obsolete comment detection and update," *IEEE Trans. Software Eng.*, vol. 49, no. 1, pp. 1–23, 2023. [Online]. Available: https://doi.org/10.1109/TSE.2021.3138909

[41] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA), 2018. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2018/summaries/627.html

[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. The Association for Computational Linguistics, 2015, pp. 1412–1421. [Online]. Available: https://doi.org/10.18653/v1/d15-1166

[44] "Word vectors for 157 languages." https://fasttext.cc/docs/en/crawl-vectors.html, 2023.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[46] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: https://dl.acm.org/doi/10.5555/2627435.2670313

[47] https://pytorch.org/.

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

[49] W. contributors., "Overlap—Wikipedia." https://en.wikipedia.org/w/index.php?title=Overlap&oldid=1061948530, 2021.

[50] F. Hassan and X. Wang, "Hirebuild: an automatic approach to history-driven repair of build scripts," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 1078–1089. [Online]. Available: https://doi.org/10.1145/3180155.3180181

[51] Y. Lou, Z. Chen, Y. Cao, D. Hao, and L. Zhang, "Understanding build issue resolution in practice: symptoms and fix patterns," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 617–628. [Online]. Available: https://doi.org/10.1145/3368089.3409760