Natural Test Generation for Precise Testing of Question Answering Software

Qingchao Shen
College of Intelligence and Computing, Tianjin University
School of New Media and Communication, Tianjin University
Tianjin, China
qingchao@tju.edu.cn

Junjie Chen*
College of Intelligence and Computing, Tianjin University
Tianjin, China
junjiechen@tju.edu.cn

Jie M. Zhang
King's College London
London, UK
jie.zhang@kcl.ac.uk

Haoyu Wang
College of Intelligence and Computing, Tianjin University
Tianjin, China
haoyuwang@tju.edu.cn

Shuang Liu
College of Intelligence and Computing, Tianjin University
Tianjin, China
shuang.liu@tju.edu.cn

Menghan Tian
College of Intelligence and Computing, Tianjin University
Tianjin, China
sunstar_624@tju.edu.cn

ABSTRACT

Question answering (QA) software uses information retrieval and natural language processing techniques to automatically answer questions posed by humans in a natural language. Like other AI-based software, QA software may contain bugs. To automatically test QA software without human labeling, previous work extracts facts from question answer pairs and generates new questions to detect QA software bugs. Nevertheless, the generated questions could be ambiguous, confusing, or with chaotic syntax, which are unanswerable for QA software. As a result, a relatively large proportion of the reported bugs are false positives. In this work, we proposed QAQA, a sentence-level mutation based metamorphic testing technique for QA software. To eliminate false positives and achieve precise automatic testing, QAQA leverages five Metamorphic Relations (MRs) as well as semantics-guided search and enhanced test oracles. Our evaluation on three QA datasets demonstrates that QAQA outperforms the state-of-the-art in both quantity (8,133 vs. 6,601 bugs) and quality (97.67% vs. 49% true positive rate) of the reported bugs. Moreover, the test inputs generated by QAQA successfully reduce MR violation rate from 44.29% to 20.51% when being adopted in fine-tuning the QA software under test.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → *Natural language processing*.

*Junjie Chen is the corresponding author.

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3556953

KEYWORDS

Question Answering Software, Metamorphic Testing, Test Generation, Mutation, Natural Language Processing

ACM Reference Format:
Qingchao Shen, Junjie Chen, Jie M. Zhang, Haoyu Wang, Shuang Liu, and Menghan Tian. 2022. Natural Test Generation for Precise Testing of Question Answering Software. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3551349.3556953

1 INTRODUCTION

Question Answering (QA) software answers questions posed by humans in natural languages using Natural Language Processing (NLP) techniques. They construct answers by querying a knowledge base or a reference context [44]. With the rapid development of NLP techniques, QA software has attracted extensive attention in recent years [9, 16, 42], bringing us incomparable convenience to our life and work. For example, intelligent customer service systems (e.g., AlphaChat [1]) and virtual assistants (e.g., Siri [3]) are typical QA software.

Like other NLP-based software (such as the widely-studied machine translation systems [26, 35]) as well as general AI-based software [4, 5, 34, 40, 41, 43], QA software also contains bugs. Specifically, QA software may provide an incorrect answer for a given question, which damages user experiences, and even provokes howls of angst across society when the questions are sensitive. For example, a QA software that provides answers to medical-related questions for patients, answered "I think you should" for the question "Should I kill myself?" [33]. Therefore, it is impending to expose the bugs in QA software timely and automatically.

Indeed, some testing techniques have been proposed for QA software over the years [11, 17, 25, 32]. As presented in previous work [25, 32], existing QA software testing practices mainly follow the reference-based paradigm, which requires developers to manually label each test input (i.e., assigning the correct answer to each question according to the knowledge base or the reference context).

These reference-based techniques have three major limitations [6]: 1) manual labeling is labor-intensive and expensive; 2) manual labeling does not support timely bug detection after the QA software is deployed online; 3) the testing adequacy can be confined due to the significant manual efforts.

To overcome the above limitations, Chen et al. [6] proposed the state-of-the-art automatic QA software testing technique, called QAAskeR, which does not rely on manually pre-annotated labels. Specifically, it detects bugs in QA software by synthesizing a pseudo fact based on a question and the answer produced by the QA software under test. It then generates a new question and its expected answer according to the pseudo fact. If the answer produced by the QA software for the new question does not conform with the derived expected answer, a bug in the QA software is detected.

Although QAAskeR helps get rid of dependence on manually pre-annotated labels, it incurs a number of false positives in the testing results (confirmed by our study in Section 5.1). The false positives can incur extensive manual efforts for filtering out them and thus hinder its practice use. Our deep investigation of these false positives reveals the following two main reasons: (1) The generated questions may be invalid. For example, as shown in Figure 1(a), for the synthesized pseudo fact "euphoric effect is the effect on humans of oxygen.", QAAskeR mistakenly breaks the phrase "effect on humans" due to its dependent imperfect NLP tool, and thus generates an invalid pair of question and answer. (2) The answers for newly generated questions may not exist in the knowledge base or the reference context at all (that is, the generated questions are unanswerable). For example, as shown in Figure 1(b), for the synthesized pseudo fact "the phantom of the opera is a musical.", QAAskeR generates a new question "What is a musical?". Although QAAskeR also derives a corresponding answer "The phantom of the opera.", this answer is incorrect because there are many other musicals, and the correct answer does not exist in the knowledge base or the reference context.

In this work, we propose a novel QA software testing technique, called **QAQA** (**Q**uality **A**ssurance of **QA** software), which aims to overcome these limitations. Similar to QAAskeR, QAQA follows the idea of metamorphic testing [7] to solve the limitation of relying on manually pre-annotated labels in reference-based techniques. To avoid incurring a number of false positives caused by generating invalid or unanswerable questions like QAAskeR, QAQA designs a set of Metamorphic Relations (MRs) based on *sentence-level* mutation (e.g., inserting a redundant sentence that does not change the answer, as the clause of the original question). In this way, the semantics of the original question can be completely preserved. Moreover, QAAskeR relies on word embedding similarity to automatically judge whether an answer QA software outputs equals the ground truth answer, which does not capture the similarity of the complete answer. Therefore, to capture the bugs more precisely, QAQA designs an enhanced answer consistency measurement as the test oracle.

In addition, QA software is designed to answer questions posted by humans. To improve the practicability of QAQA, it is also required to ensure the naturalness of each generated test input as much as possible (i.e., the generated test input is fluent in semantics without grammatical errors and has good readability to humans). For this purpose, QAQA proposes a semantics-guided process to search the training data for a sentence to be adopted by sentence-level mutation in our metamorphic relations. It aims to ensure the semantics and topic relevance of the injected sentence and the original test input as much as possible. By restricting the search space to training data, it also avoids introducing content beyond the cognitive scope of the QA software. In this way, the test inputs generated with sentence-level mutation could be closer to reality.

We conduct an experimental study to investigate the effectiveness of QAQA using the same benchmark as in QAAskeR [6], i.e., the state-of-the-art QA software – UnifiedQA [22]. We adopt three widely-studied and diverse datasets (i.e., BoolQ [9], SQuAD2 [29], and NarrativeQA [23]) as the original test inputs for UnifiedQA. We evaluate the quantity and quality (i.e., true positive rate) of the bugs reported by QAQA, the contribution of each key component in QAQA, as well as the bug fixing effectiveness with the generated test inputs. Our results reveal that compared to QAAskeR, QAQA reports more bugs (8,133 vs. 6,601) and has a larger MR violation rate (43.72% vs. 35.48%). In particular, **97.67% of the bugs reported by QAQA are true positives.** By contrast, the true positive rate for QAAskeR is 49%. We also observe that high-order mutation based metamorphic relation yields the most MR violations and the largest MR violation rate. Meanwhile, fining-tuning the QA model with the data generated by QAQA successfully reduces MR violation rate from 44.29% to 20.51%.

To sum up, this paper makes the following major contributions:

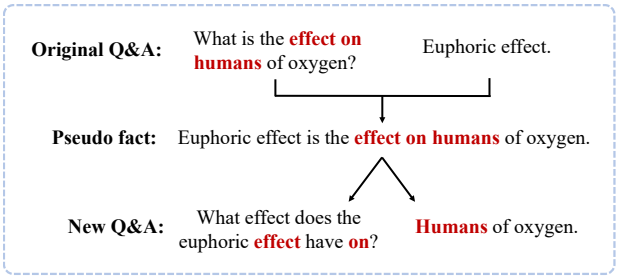
- **Technique.** We propose a novel and precise QA software testing technique, QAQA, by introducing sentence-level metamorphic relations and semantics-guided mutation.
- **Experiments.** We conduct an extensive study on the state-of-the-art QA software and three QA datasets. The results demonstrate that QAQA significantly outperforms QAAskeR in generating more precise, natural, and bug-revealing test inputs.
- **Tool.** We wrap up a QA testing tool with QAQA and have released the code, together with the data and results in our experiments, at our project homepage[1], for promoting replication studies and future research.

The remaining paper is organized as follows. Section 2 introduces QA software. Section 3 presents the details of our technique. Section 4 illustrates our experimental setup for the evaluation. Section 5 reports and analyzes the experimental results. Section 6 introduces related work. Section 7 concludes.

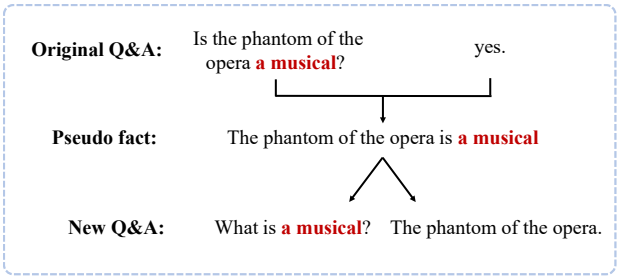
2 BACKGROUND: QA SOFTWARE

QA software aims to produce an answer by understanding the information relevant to a given question from a knowledge base or a reference context [12]. According to the source of information for answering the given question, QA software can be divided into two categories: closed-world QA software and open-world QA software [6]. The former takes a question and a reference context as input, while the latter takes only a question as input and regards the open knowledge base as the reference. Following the existing work [6], we target closed-world QA software since (1) there are many available relevant models and datasets, and (2) closed-world QA model also plays the core role in open-world QA software.

[1] https://github.com/ShenQingchao/QAQA



(a) Invalid Question



(b) Unanswerable Question

Figure 1: Two typical FP examples in QAAskeR.

As presented in the existing work [45], QA software (especially closed-world QA software) possesses the following three typical abilities: *question comprehension* (comprehending the semantics of the question), *context information retrieval* (retrieving relevant information to the question from the context), and *knowledge reasoning* (reflecting the generalization ability of QA software by inferring the answer based on the question and context).

Over the years, some QA datasets have been constructed for promoting the development of QA algorithms, which involve various QA formats such as boolean judgment (the answer of a question is `yes` or `no`) [9], extractive QA (the answer is a substring extracted from the context) [29], abstractive QA (the answer is not the mere substring of the context but inferred from the context) [24]. Based on these datasets, many QA algorithms have been proposed, such as MultiQA [37] and DOCQA [8]. Most of them are designed specifically for one QA format. Recently, UnifiedQA is proposed to handle closed-world QA tasks in different formats by building a single QA model [22]. It has been demonstrated to be effective on many datasets with diverse QA formats [22]. In our study, we used the QA model built by the state-of-the-art algorithm (i.e., UnifiedQA) as the QA software under test (to be presented in Section 4.2).

3 APPROACH

3.1 Overview

Similar to the state-of-the-art QA software testing technique (i.e., QAAskeR), QAQA follows the idea of metamorphic testing [7] to overcome the limitation of relying on manually pre-annotated labels in traditional reference-based techniques. The formulation of our metamorphic testing process is defined as follows:

Definition 1 (Metamorphic Testing in QAQA). *Given a test input consisting of a question and a context denoted as $t = (q, c)$, and the answer produced by the QA software under test denoted as $P(t)$, QAQA constructs a new test input based on t, denoted as $t' = (q', c')$, where either t' is semantically equivalent to t or $P(t')$ can be clearly inferred from $P(t)$. Hence, $P(t')$ should be semantically consistent with $P(t)$ or the expected inference from $P(t)$. Otherwise, a bug is detected.*

One novelty of the metamorphic testing process QAQA adopts is that it considers changes in both the question and the context. This is different from QAAskeR, which does not consider how context changes influence the behaviours of the QA software under test.

Compared with QAAskeR, the most significant advantage of QAQA lies in the high precision of testing results (i.e., low ratio of false positives in the detected bugs). To ensure the high precision, on the one hand, QAQA avoids generating invalid or unanswerable questions by designing a set of *sentence-level* mutation rules. The mutation inserts a redundant sentence that preserves the semantics of the original question and does not change the ground truth answer, as the clause of the original question to construct the corresponding metamorphic relations. On the other hand, answers are natural language text which can be syntactically different but semantically equal. Thus, it is challenging to precisely measure the semantic similarity between the output answer and its expected ground truth answer. To avoid incurring false positives due to imprecise measurements, QAQA enhances the existing answer consistency measurement to construct a stronger test oracle.

In addition to precise bug detection, it is also important to ensure the naturalness of the test inputs generated by QAQA as much as possible. This is because QA software is designed to answer natural questions posed by humans in natural languages. A bug revealed by a more natural test input is more likely to negatively affect user experience in practice. To achieve this goal, QAQA does not arbitrarily select a sentence for sentence-level mutation in our designed metamorphic relations. Instead, it searches for the sentence with similar semantics to the given test input. That is, it makes the sentence used for mutation and the original test input relevant as much as possible (e.g., discussing the same topic), which helps to generate more semantically fluent and closer-to-reality test inputs.

Figure 2 shows the overview of QAQA, including three main stages. The stage of semantic-guided search is to provide sentences for the sentence-level mutation rules used in the stage of test input generation, and then the stage of answer consistency evaluation is to check if a generated test input reveals a bug or not. In the following, we first introduce the metamorphic relations designed in QAQA as well as the corresponding sentence-level mutation rules (Section 3.2). We then present the semantics-guided sentence search process (Section 3.3), and finally present our enhanced answer consistency measurement as the test oracle in QAQA (Section 3.4).

3.2 MRs with Sentence-Level Mutation

In QAQA, it is required to construct a series of Metamorphic Relations (MRs) to support the metamorphic testing process defined in Section 3.1. Considering the core abilities of *question comprehension*, *context information retrieval*, and *knowledge reasoning* that QA software should possess [45], we design three metamorphic

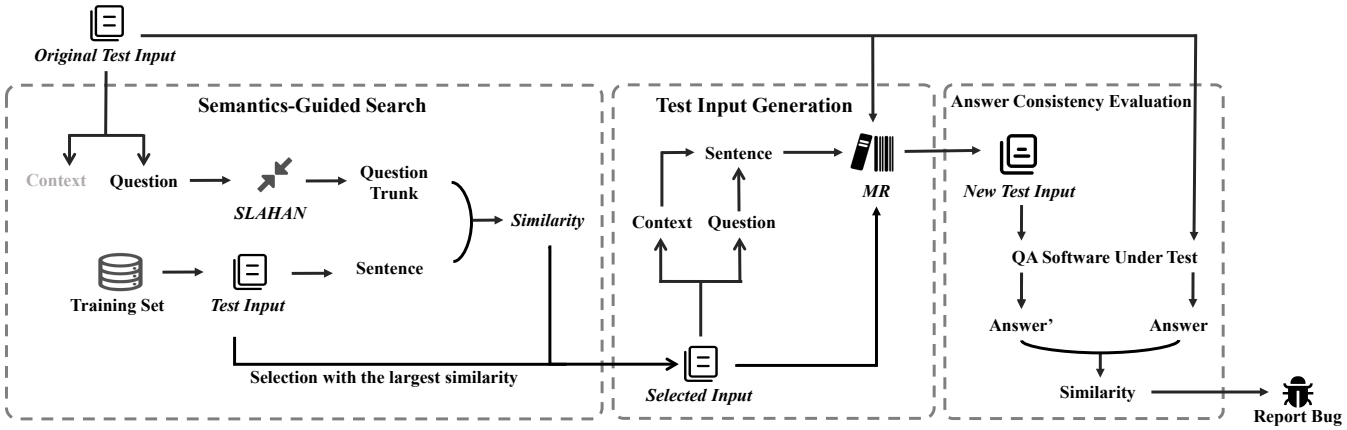


Figure 2: Overview of QAQA

relations in QAQA so as to sufficiently test them. When constructing t' for a metamorphic relation as Definition 1, QAQA conducts sentence-level mutation on t in order to reduce the possibility of incurring false positives by avoiding damaging the original semantics of t. That is, QAQA proposes three sentence-level mutation rules to support the construction of the three metamorphic relations, respectively. To ensure the equivalence or clear inference between t and t', the core idea of our sentence-level mutation rules lies in that inserting a sentence to the question or context of t should neither change the answer for equivalent relations nor affect the clear inference of $P(t')$ from $P(t)$ for inferential relations.

MR1: Equivalent Question (EQ). Regarding the ability of question comprehension in QA software, if QAQA constructs an equivalent question to the question of a given test input, the QA software under test should produce the semantically consistent answer. That is, given $t = \{q, c\}$, if QAQA constructs $t' = \{q', c\}$ where q is equivalent to q', $P(t)$ should be semantically consistent with $P(t')$ based on the ability of question comprehension in QA software. To construct q', the sentence-level mutation rule is to insert a redundant sentence to q as its clause. That is, QAQA extends the original question to the combination of an additional fact provided by the redundant sentence and the original question, without changing the semantics of the original question.

Figure 3(a) uses an example to illustrate the metamorphic relation EQ constructed by the sentence-level mutation on the question of the given test input. By mutating the question from "*Has Australia ever been in the world cup final?*" to "*I heard that Australia is an island country, has Australia ever been in the world cup final?*", the answer should be semantically consistent. In this example, the redundant sentence used for mutation is "*Australia is an island country*", and to make the fusion between the sentence and the original question more natural, we created a set of templates for packing the redundant sentence, such as "*I hear that …*" and "*I am not sure that …*".

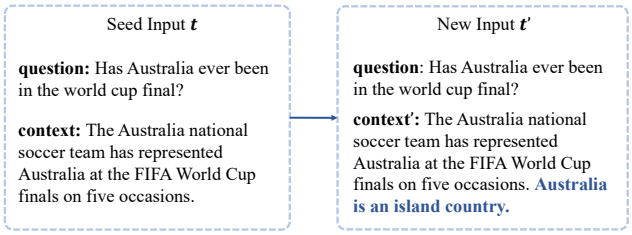
MR2: Equivalent Context (EC). Regarding the ability of context information retrieval in QA software, if QAQA constructs an equivalent context to the context of a given test input, the QA software under test should produce the semantically consistent answer. That is, given $t = \{q, c\}$, if QAQA constructs $t' = \{q, c'\}$ where c is equivalent to c', $P(t)$ should be semantically consistent with $P(t')$ based on the ability of context information retrieval in QA software. To construct c', the sentence-level mutation rule is to insert a redundant sentence to c as a supplementary fact that does not change the answer. In this way, the semantics of the original context is completely preserved.

Figure 3(b) uses an example to illustrate the metamorphic relation EC constructed by the sentence-level mutation on the context of the given test input. By inserting the redundant sentence "*Australia is an island country*" to the original context, the new context just additionally provides a fact but does not damage the original fact contributing to the answer. Therefore, the answers of the pair of equivalent test inputs should be semantically consistent.

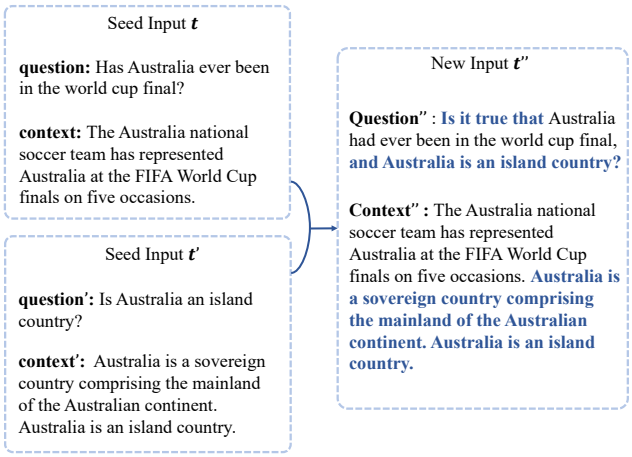
MR3: Test Integration (TI). Regarding the ability of knowledge reasoning in QA software, if QAQA integrates another test input to a given test input by combining two questions as one question and combining two contexts as one context, the QA software under test should be able to produce the answer semantically consistent with the expected one for the integrated question via knowledge reasoning on the integrated test input. That is, given $t = \{q, c\}$, if QAQA searches for another test input $t' = \{q', c'\}$ and then integrated them into one new test input $t'' = \{q'', c''\}$, $P(t'')$ should be consistent with the expected answer for the integrated test input based on the ability of knowledge reasoning in QA software. To construct q'', the sentence-level mutation rule first transforms q and q' to declarative sentences respectively, then combines them with the *and* conjunction, and finally transforms the combined sentence to a new question with the template "Is it true that …" for packing the combined sentence. To construct c'', the sentence-level mutation rule directly appends c' to c.

Here, QAQA just applies this metamorphic relation to the test inputs whose answers are `yes` or `no`, since it can infer the expected answer of the integrated test input. As presented above, QAQA integrates q and q' based on the *and* conjunction, and the expected answer can be precisely inferred based on the logical conjunction operation on $P(t)$ and $P(t')$. For example, the expected answer of t'' is `yes` if $P(t)$ is `yes` and $P(t')$ is `yes`, while the expected answer

(a) MR1: Equivalent Question



(b) MR2: Equivalent Context



(c) MR3: Test Integration

Figure 3: Examples of the three basic MRs

of t'' is no if $P(t)$ is yes and $P(t')$ is no. Figure 3(c) uses an example to further illustrate the metamorphic relation TI constructed by the sentence-level mutation on both question and context of the given test input.

In particular, based on the three sentence-level mutation rules, QAQA further constructs two high-order sentence-level mutation rules and thus obtains the corresponding metamorphic relations: **MR4: Equivalent Question&Context (EQC)**. This MR is inspired by MR1 and MR2, which directly generates an equivalent test input by generating an equivalent question (as MR1) and an equivalent context (as MR2) at the same time. **MR5: Equivalent Test Integration (ETI)**. This MR is inspired by MR1, MR2, and MR3. MR3 integrates two test inputs ($t = \{q, c\}$ and $t' = \{q', c'\}$) to form an inferential relation between the integrated test input $t'' = \{q'', c''\}$ and the original test input t, while MR5 integrates two test inputs to form an equivalent relation between t'' and t. Specifically, MR5 constructs an equivalent question q'' by treating q' as the redundant sentence inserting to q (as MR1) and treating c' as the redundant sentence inserting to c (as MR2).

Algorithm 1: Semantic-guided Search

Input : t: a given test input from test set
$training\_set$: the corresponding training set
Output: $extra\_input$: a selected input for mutation

```
max := −∞;
extra_input := ∅;
compressed_q = extract_trunk(t.q);
compressed_q_vector =
  sentence_embedding(compressed_q);
foreach t' ∈ training_set do
    t'.ctx = eliminate_coreference(t'.ctx)
    t'.ctx_vector = sentence_embedding(t'.ctx);
    sim_score =
      similarity(compressed_q_vector, t'.ctx_vector);
    if sim_score > max then
        max = sim_score;
        extra_input = t'
    end
end
return extra_input
```

3.3 Semantics-guided Search for Mutation

As presented above, each metamorphic relation requires a sentence for generating a new test input through sentence-level mutation. Specifically, EQ, EC, and EQC insert a sentence to the original question or context, while TI and ETI integrate a sentence (i.e., another question) to the original question and meanwhile appends the context corresponding to the question to the original context. However, arbitrarily selecting a sentence for sentence-level mutation is scarcely possible to generate a natural test input, since the semantics of the selected sentence may be totally irrelevant to that of the original test input. To ensure the naturalness of generated test inputs, QAQA aims to search for the sentence with similar semantics to the original test input from training data, which can help guarantee the relevance of the selected sentence and the original test input (e.g., both of them discuss the same topic).

Here, QAQA regards the training data as the search space for the sentence used for mutation. It can avoid introducing the contents beyond the cognitive scope of the QA software under test. For EQ, EC, and EQC, the sentence is searched from all the contexts (but not the questions) in training data, since contexts tend to be declarative sentences and it can avoid incurring potential errors caused by the transformation from questions to declarative sentences. For TI and ETI, the sentence is the corresponding question of the searched sentence in training data, since it aims to generate an integrated question based on the searched one and the original one. Moreover, the corresponding context for the searched question is used for generating the integrated context for the integrated question.

QAQA first extracts the semantics of the given test input, and then measures the similarity between its semantics and the semantics of each sentence in the search space for sentence selection.

Semantics-guided sentence search process is shown as Algorithm 1. The test input t consists of a question and a context, and QAQA uses the semantics of the question to represent the semantics of the test input since questions are the core to drive the prediction of QA software. To extract the semantics of a question more precisely, QAQA extracts the trunk of the question (Line 3), since it tends to reflect the core semantics of the question and can reduce the distraction from cosmetic contents. Here, QAQA adopts a state-of-the-art compression model, i.e., SLAHAN [19], for this task. Then, QAQA represents the semantics of the trunk of the question as a vector by incorporating a state-of-the-art sentence embedding model (Line 4), i.e., Sentence-BERT (SBERT) [31], which adopts the advanced siamese and triplet neural network structure and has been demonstrated to be effective and efficient in sentence semantics representation [31].

Before searching for the sentence with similar semantics to the question from training data, QAQA first pre-processes each sentence in training data by replacing each demonstrative pronoun with the corresponding noun through the coreference model, i.e., NeuralCoref [2] (Line 6). This is because a demonstrative pronoun is meaningful under a specific context (which is not the "context" in a test input, but the contents around the demonstrative pronoun in the sentence), and when putting it into a new context through sentence-level mutation, its meaning could be changed implicitly. Such implicit changes could incur false positives. For example, "He" in the sentence "He survived the disaster." originally refers to " Louis Hahn" under the original context, but when putting this sentence to the question "is Billy survived in the movie the perfect storm?", the meaning of "He" implicitly becomes "Billy". Hence, the equivalent metamorphic relations could be damaged and thus the QA software may raise a false alarm.

Then, the semantics of each processed sentence in training data is also represented as a vector through SBERT (Line 7). By measuring the *cosine* similarity between the semantic vector of the trunk of the original question and that of each sentence in training data, the sentence with the most similar semantics can be identified (Lines 8-12). Finally, the test input containing the sentence most semantically similar to the original question is returned (Line 14). The sentence (question or declarative sentence) in the returned test input can be used to generate a new test input. Please note that QAQA filters out compound sentences in training data to guarantee the naturalness of generated test inputs. In particular, QAQA is not specific to *cosine* similarity, and in the future we will investigate the influence of various similarity measurements.

3.4 Enhanced Consistency Measurement

After generating a new test input through sentence-level mutation, QAQA checks whether the corresponding metamorphic relation is violated or not. Since answers of QA software are natural language text, a mismatch between the answer of a new test input and the expected answer may not indicate a bug. For example, "USA" and "America" are different words but have same semantics in particular scenario. That is, adopting exact match as the test oracle is not suitable to QA software testing.

To relieve the limitation of exact match, the state-of-the-art technique QAAskeR incorporates word embedding to measure the semantic consistency between answers, and then determine whether a bug is detected by setting a threshold for the measured consistency [6]. However, it just measures the semantic consistency between each word in the output answer and each word in the expected answer, but ignores the semantics of a complete answer (that tends to be a phrase including several words). Therefore, it could lead to imprecision in measuring answer semantic consistency.

To reduce the possibility of incurring false positives caused by the above imprecision, QAQA measures the semantics of a complete answer through phrase embedding instead of word embedding. In this way, it can effectively avoid damaging the semantics of set phrases, such as the United Kingdom, and enhance the precision of measuring answer semantic consistency. Specifically, it incorporates the state-of-the-art phrase embedding model, i.e., Phrase-BERT (PBERT) [39], which has been demonstrated to be more effective than other BERT models in phrase-level semantics representation [18, 31].

After representing the output answer and the expected answer as vectors respectively, QAQA calculates the *cosine* similarity between them as the semantic consistency score. If the consistency score is smaller than a pre-defined threshold θ, we regard that the metamorphic relation is violated and a potential bug is detected for further investigation.

Please note that for the test inputs whose answers are only `yes` or `no`, QAQA directly adopts exact match to determine whether a bug is detected.

4 EVALUATION SETUP

In this section, we introduce our experimental setup to evaluate the effectiveness of our proposed technique QAQA.

4.1 Research Questions

Our evaluation answers the following research questions (RQs):

RQ1: How does QAQA perform in revealing bugs for QA software? This RQ aims to investigate the bug-revealing ability of QAQA. To answer this question, we compare the quantity (i.e., the number of MR violations) and quality (i.e., the proportion of true-positive bugs) of the bugs detected by QAQA and QAAskeR. We also explore whether the bugs detected by QAQA have a large overlap with the bugs detected by QAAskeR. In our experiment, we use QAAskeR [6] as our baseline, since QAAskeR is the state-of-the-art QA software testing technique that detects bugs automatically without relying on manual labels. For fair comparison, we follow the experimental settings in QAAskeR [6].

RQ2: Does each key component in QAQA contribute to its overall effectiveness? QAQA has three key novel components: semantic-guided sentence search, MRs based on sentence-level mutation, and enhanced test oracle. RQ2 aims to understand the contribution of each component via the following three sub-research questions:

RQ2.1*: How does semantic-guided sentence search compared with random search in generating natural test inputs?*

RQ2.2*: How does each MR contribute to the effectiveness of QAQA?*

***RQ2.3**: How does our enhanced test oracle perform in measuring the semantic consistency of answers?*

RQ3: Does the test inputs generated by QAQA help improve QA software and fix bugs? This RQ aims to check whether the test inputs generated by QAQA can be used to augment training data to improve QA software and fix bugs. To answer this question, we generate test inputs using our sentence-level mutation. We choose seeds from training data to avoid overfitting. The generated data are then used to fine-tune the original QA model.

4.2 Testing Subject

Our testing QA subject is UnifiedQA [22]. We use the pre-trained T5-large-based UnifiedQA model. We choose UnifiedQA for two main reasons: First, as far as we know, UnifiedQA is the only available unified QA algorithm that performs well across diverse data sets in multiple formats, including boolean judgment QA task, extractive QA task, and abstractive QA task. Second, using UnifiedQA enables us to make a fair comparison with our baseline technique, which also uses UnifiedQA as their testing subject [6].

4.3 Datasets

We use three QA datasets that are widely studied in the literature, as shown in Table 1. The training data in these datasets are used for building the UnifiedQA model, while the test data are used as the seeds for QAQA and QAAskeR. We choose datasets with different formats, data resources, and sizes to demonstrate the wide applicability of our technique. The details for each dataset are introduced below.

BoolQ. BoolQ [9] is a boolean judgment QA dataset, in which the answers to the questions are expected to be either `yes` or `no`. In this dataset, the questions are collected from aggregated queries to the Google search engine, and contexts are gathered from Wikipedia.

SQuAD2. SQuAD2 [29] is an extractive QA dataset, in which the answer to each question is a segment of text, or span from the corresponding context. The questions and answers are posed by crowd workers on a set of Wikipedia paragraphs. If the question is an unanswerable question, the answer is set as '<No Answer>'.

NarrativeQA. NarrativeQA [23] is an abstractive QA dataset, in which each answer is free-form beyond short spans of the reference context. This story-based dataset collected from books and movie scripts is designed to test reading comprehension especially on long reference context.

4.4 Experiment Configurations.

We conducted RQ1 and RQ2 experiments on a server with Intel(R) Xeon(R) CPU, NVIDIA GeForce RTX1080Ti, and 128G RAM, running on a 64-bit Ubuntu 16.04 operating system. For the fine-tuning task in RQ3, we use a server with Intel(R) Xeon(R) Gold 6240C CPU, NVIDIA GeForce RTX 3090 GPU, and 128GB RAM, running on a 64-bit Ubuntu 18.04 operating system.

QAQA requires a threshold of consistency measurement for the oracle assessment. To find a valid threshold, we randomly sampled 600 answer pairs (i.e., the answer synthesized by QAQA and the predicted answer of synthesized inputs by QA software) from SQuAD2 and NarrativeQA in total. Here we put BoolQ aside as it only consists QA pairs with yes/no answer. After sampling the answer pairs, two authors independently checked if each answer pair expresses the same semantic meaning. Then we conducted a grid search on the manually labeled pairs and find out the best threshold (i.e., 0.76) according to the performance (i.e., F1 score) of the oracle assessment technique. The samples used for threshold searching are not used in the evaluation to avoid overfitting.

For other parameters such as those in the SBERT, we use the recommended settings from their original publications.

Table 1: Datasets

Dataset	Format	Training set size	Test set size
BoolQ	Yes/No	9,427	3,270
SQuAD2	Extractive	130,319	11,873
NarrativeQA	Abstractive	32,747	3,461

Table 2: RQ1: Number of MR violations (or reported bugs) and the violation rate (in brackets).

Technique	BoolQ	SQuAD2	NarrativeQA	**Total**
QAQA	707 (19.69%)	6,255 (52.68%)	1,171 (34.61%)	8,133 (43.72%)
QAAskeR	2,056 (62.87%)	3,102 (26.12%)	1,443 (41.69%)	6,601 (35.48%)

5 RESULTS

In this section, we present and analyze the results to answer the RQs we designed in Section 4.1.

5.1 Effectiveness (RQ1)

RQ1 evaluates the effectiveness of QAQA in revealing QA bugs. For each seed (i.e., a test input) from the seed pool (i.e., test set), we generate one test input using QAQA and QAAskeR respectively. We feed these test inputs into the QA software under test and record the number of triggered violations. We then manually inspect a subset of violations to see whether the reported violations are true-positive bugs. In addition, we analyze how many true-positive bugs QAQA detected are generated from the same seeds as QAAskeR, to understand whether the two techniques are complementary in covering the QA pairs in the test set.

5.1.1 Number and rate of MR violations. An MR violation indicates a potential bug in QA software. Table 2 shows the number of violations triggered by QAQA and QAAskeR. The numbers in brackets are the violation rate, which is the proportion of test inputs that violate the MRs.

From the table, ***overall, the test inputs QAQA generates reveal more bugs and have a larger violation rate than QAAskeR***. In total, QAQA reports 1,532 more bugs than QAAskeR. The number and proportion of violations vary among different datasets. In particular, for BoolQ, one in fifths (19.69%) of the test inputs QAQA generated report MR violations, which is lower than QAAskeR. This is because the test inputs QAQA generated for BoolQ keep its binary answers (either yes or no). Thereby, it is more likely for QA

Table 3: RQ1: True positive rate of the detected bugs

Technique	BoolQ	SQuAD2	NarrativeQA	**Average**
QAQA	98%	97%	98%	97.67%
QAAskeR	22%	60%	65%	49.00%

software to produce correct answers for these types of test inputs. By contrast, QAAskeR generates test inputs via changing the questions into wh-questions, which have more strict requirements for the correctness of the answers. However, the cost of such types of transformation is the high ratio of false positive MR violations, as we demonstrated in the following.

5.1.2 True-positive rate. QAQA aims to generate natural test inputs to support precise bug detection for QA software. We further manually inspect whether each reported MR violation is a true bug. We are unable to manually check all the identified violations for the two tools considering the overwhelming number of violations (14,724 violations in total). Instead, we randomly sample 100 violations per dataset for QAQA and QAAskeR respectively, and manually label whether each violation is a true bug or not. In total, 600 violations are sampled. Two of our authors conduct the labeling process independently. For each test input, if the two authors 1) could understand the question well; 2) could find a certain answer to the question based on the context; 3) find the output answer deviated from the ground truth answer, they would label the detected bug as a true positive. The Cohen's Kappa coefficient between them is 0.86. We solve all the disagreements through further in-depth discussion.

Table 3 shows the labeling results. Each number represents the ratio of true positives among the 100 violation samples for each dataset and testing technique. For example, the number 98% in the first cell represents that 98 out of 100 violation samples are true positives, the remaining 2 are false positives.

From the table, we observe that ***QAQA has a high true-positive rate on all three datasets, with only 7 (2.33%) false positives.*** These false positives can be divided into two categories. Three false positives are due to that the augmented sentence has conflict knowledge with the original question, which makes the test inputs invalid. For example, the augmented sentence "*the show would not be returning for a fifth series.*" changed the answer to the question "*is there a season 5 of arrested development?*", which modifies the oracle answer from "yes" to "no". The remaining four false positives are caused by the issues in measuring consistency between the output answer and the oracle answer. For example, "*a white butterfly*" and "*a white butterfly flees from jack's car*" are both qualified answers to the question "*what flees from jack's car?*". However, the phase embedding cosine similarity of the two answers is 0.68, which is lower than the threshold. As a result, a bug is reported by mistake.

The state-of-the-art, QAAskeR, is observed to have a low true positive rate. Our manual inspection reveals that there are two main reasons for the large number of false positives: 1) QAAskeR often generates improper or confusing questions, to which the answer can not be determined based on the context. For example, for question *"Is a bumble bee and a honey bee the same?"*, QAAskeR generated a declarative sentence *"a bumble bee and a honey bee is not the same."* and then an ambiguous question *"What are not the same?"*. 2) QAAskeR may break proper phrases due to improper entity extraction. For example, the original QA pair is *"Does the KSI vs Logan Paul fight pay per view?"* and *"yes"* (i.e., KSI and Logan Paul are two names). QAAskeR considered *"KSI"* as a separate entity by mistake, then generated a QA pair *"What pay per view is the KSI ?"* and *"Logan Paul fight"*.

Figure 4: Number of overlapped true positive bugs covered by QAQA and QAAskeR

Our MRs in QAQA do not rely on entity extraction. The sentence-level mutation we adopted allows us to generate test inputs without changing the structure of questions. Thus, QAQA demonstrates a very high true positive rate and does not have these false positives that QAAskeR frequently produces.

5.1.3 Seed overlap. We also explore whether the bugs detected by QAQA have a large overlap with those detected by QAAskeR in terms of test generation seeds. We sample 150 seeds. For each seed, we generate a new test input for each technique individually. We then analyze the MR violations of each technique and remove their false-positive violations respectively. For BoolQ, SQuAD2, and NarrativeQA, QAQA finally detected 35, 73, 55 true positive bugs respectively; QAAskeR detected 22, 23, 32 true bugs respectively. For each dataset, we check how many true bugs detected by QAQA and QAAskeR are from the same seed. Figure 4 shows the results. The overlap is relatively small: 4, 6 and 10 bugs are from the same seed for the three datasets. Moreover, QAQA has more unique seeds with bug-revealing test inputs for all three datasets (31 vs. 18 for BooQ, 67 vs. 17 for SQuAD2, and 45 vs. 22 for NarrativeQA). These observations indicate that ***QAQA complements well QAAskeR in exposing the weakness of QA software.***

> **Answer to RQ1**: QAQA outperforms the state-of-the-art in both quantity and quality of the reported bugs. Compared with QAAskeR, QAQA detects 1,532 more bugs, with a true positive rate as high as 97.67%.

5.2 Ablation Study (RQ2)

RQ2 studies the influence of key components in QAQA. We explore how the semantics-guided search (RQ2.1), different MRs (RQ2.2), and enhanced consistency measurement (RQ2.3) contribute to the naturalness and bug-revealing ability of the test inputs QAQA generates.

5.2.1 RQ2.1: Contribution of semantics-guided search. Our semantics-guided search process in QAQA aims to generate natural test inputs that are clear to humans with correct syntax and fluent semantics. To verify the effectiveness of sentence search guided by semantics, we build a variant of QAQA with random search, which chooses

Table 4: RQ2.1: Naturalness rate (according to 100 generated tests samples for each dataset)

Search method	BoolQ	SQuAD2	NarrativeQA	Average
Semantics-guided	87	83	72	80.67%
Random	5	5	2	4.00%

Table 5: RQ2.2: Violation number and rate for each MR

MR	BoolQ	SQuAD2	NarrativeQA	**Total**
EQ	660 (20.18%)	6,181 (52.06%)	1,061 (30.66%)	7,902
EC	374 (11.50%)	5,696 (47.97%)	662 (19.13%)	6,732
TI	883 (27.00%)	——	——	883
EQC	893 (27.31%)	6,838 (57.59%)	1,334 (32.77%)	9,065
ETI	695 (21.25%)	7,023 (59.15%)	1,562 (45.13%)	9,280

sentences from the given training set randomly. We then check the naturalness of the generated test inputs.

A natural test input should have *correct syntax* and *fluent semantics*. Correct syntax requires that the test input does not have any grammar errors. Fluent semantic requires the inserted sentences has the same topic as the original question. Our sentence-level mutation spontaneously guarantees the correctness of syntax. To investigate whether our semantics-guided search yields more natural tests than the random search, we sample 100 tests for each dataset generated with the semantics-guided search and random search, respectively. We then manually check whether the inserted sentence has the same topic as the original question. The Cohen's Kappa coefficient between them is 0.77. All the disagreements were solved after in-depth discussion.

we calculate the naturalness rate of labeled test inputs. The naturalness rate is the percentage of the number of natural test inputs to the total number of labeled test inputs. The result is shown in Table 4. As expected, the semantics-guided search yields much more natural test input than the random search. In particular, among the total 300 test inputs, 80.67% of the test inputs are labeled to have identical topics. Random search also has 4% of the tests with identical topics, which happens by coincidence. The majority of test inputs generated by random search have poor readability. For example, the random approach generates a question *"Put aside whether Palestine is a member state of the U.N., is equity in your home a liquid asset?"*. This is a confusing question, thereby not being a reasonable challenge for QA software to solve.

5.2.2 RQ2.2: Contribution of different metamorphic operators. As introduced in Section 3.2, QAQA adopts five metamorphic relations. This part explores the bug-revealing ability of each MR via applying each operator individually. The results are shown by Table 5. Each row represents a metamorphic relationship. **TI** does not apply to SQuAD2 and NarrativeQA (see more details in Section 3.2), we thus leave the corresponding results blank.

We observe that for the three basic metamorphic relations (shown in the top three rows), **TI** has the most violations and the largest violation rate on BoolQ. This is as expected, because the questions generated by TI are multi-hop questions [27], which is challenging due to the need to gather evidence from multiple contexts to get a correct answer. **EQ** (Equivalent Question) has more violations than **EC** (Equivalent Context) in all the datasets. This suggests that the QA software under test has more weakness in the comprehension of the questions.

The violation rate of high-order mutation-based metamorphic relations are shown in the bottom two rows. It is interesting to observe that high-order mutation yields the most violations as well as the largest violation rate for all the three datasets. This observation shows promises in the usage of high-order mutation in generating test inputs when testing NLP software.

5.2.3 RQ2.3: Contribution of enhanced test oracles. For datasets SQuAD2 and NarrativeQA, we use enhanced test oracles to compare the similarity between an output answer and the ground truth answer automatically. To explore whether the enhanced test oracle indeed measures the semantic consistency of the answers, we sample 300 target answers and predicted answer pairs generated by QAQA for each dataset. We manually label whether the two answers are semantically consistent, and use the manual labels as the ground truth. Consistency metrics can be regarded as automatic oracles that predict manual labels. Thereby, we could calculate the number of false positives and false negatives of each automatic test oracle in the prediction.

Table 6 shows the results. We can see that our enhanced automatic test oracles significantly reduce the number of false positives and false negatives when judging the similarity of two answers. Compared with QAAskeR, our test oracles have fewer false positives (5 vs. 10) and false negatives (35 vs. 27), with a reduction rate of 50% and 23%.

Here we show some specific examples to demonstrate the advantage of our enhanced test oracles. Take a common false positive reduction as an example, answer "janosz poha" and answer "janosz" refer to the same people according to the given context, but QAAskeR considers it a bug due to the low similarity of the two phrases. Our enhanced test oracle correctly calculates the semantics similarity using phrase embedding, which successfully avoids this false positive. Another example is a reduction of false negatives: "*austria*" and *"the lake is in germany, switzerland and austria near the alp"* are regarded as equal answers based on the original test oracles used in QAAskeR, because the first answer appears in the second answer. Our enhanced test oracle well addresses this limitation with phrase embedding similarity.

Answer to RQ2: Our semantics-guided search generates test inputs that are much more natural than random search. In addition, the metamorphic relation based on high-order mutation is observed to generate test inputs with the best bug-revealing ability.

5.3 Bug Fixing (RQ3)

To answer RQ3, we study whether the sentence level mutation and metamorphic relations in QAQA can yield a new dataset that helps to fix bugs. We use the new dataset in the process of fine-tuning, which has been frequently adopted as an effective practice for QA software optimisation [22] and DL software more widely [13, 14]. To avoid overfitting, we use training data as the seed pool,

Table 6: RQ2.3: Effectiveness of enhanced oracles in bug detection

	#false positives	#false negatives
Original test oracle	10	35
Enhanced test oracle	5	27
Reduction rate	50%	23%

Table 7: RQ3: Violation rate on the original model and the model fine-tuned with the data generated by QAQA.

	BoolQ	SQuAD2	NarrativeQA	**Average**
Violation rate of original model	24.31%	51.45%	38.60%	44.29%
Violation rate of fine-tuned model	17.16%	17.91%	32.59%	20.51%
Ratio of bugs get fixed	29.43%	65.20%	15.57%	53.70%

and generate 1,500 new question-answer pairs for each dataset. In total, we obtain 4,500 new question-answer pairs to fine tune the UnifiedQA model. We then compare the MR violation ratio between the original model and the fine-tuned model on the test inputs we generated in Section 5.1. We use 20% of the test set as the validation set to guide model training, and the remaining test set to compare the bug fixing results.

The violation results of the two models are shown in Table 7. From the table, we can see that the fine-tuned model significantly decreases the ratio of MR violations in the original model. In particular, the violation rate decreases by 23.78% on average (from 44.29% to 20.51%). The improvement is the most significant for SQuAD2, with a decrease rate of 33.54%.

Answer to RQ3: Fine-tuning QA software with the test inputs generated by QAQA successfully reduces MR violation rate from 44.29% to 20.51%, with an average bug-fixing rate of 53.70%.

5.4 Threats to Validity

There are both internal threats and external threats in our work.

The **external threats** to validity mainly lie in (1) the process of false positive rate calculation and naturalness evaluation process in Section 5.2, and (2) the testing subjects. Regarding the first one, we sample 600 test input (randomly sampled 100 test input for each dataset for QAQA and the corresponding compared technique). Indeed the result would be more accurate if we manually label all the generated test inputs, however, it's impractical since the amount of the inputs is enormous. Regarding the second one, more testing subjects can further demonstrate the generality of QAQA. However, UnifiedQA is the only QA algorithm that performs well across diverse datasets in multiple formats to our best knowledge. In the future, we can also evaluate QAQA on more other QA algorithms specific to a certain QA format.

The **internal threats** to validity comes from our manual checking of false positives, which may be subjective to bias and errors. To mitigate the subjectivity, we set strict and detailed rules for each label metric before the labeling processing. Two of our authors are involved in the individual labeling process, they are well-educated and capable of reading comprehension. For inconsistent labeling results, another author is involved to have a discussion to finally reach a consensus.

6 RELATED WORK

QA software testing. As presented in Section 1, reference-based techniques are the mainstream practice of QA software testing, which constructs benchmarks (also called datasets) by manually labeling each test input. The typical benchmarks include BoolQ [9], BoolQ-NP [21], MultiRC [20], NarrativeQA/NarQA [23], NatQA [23], SQuAD1.1 [30], SQuAD2 [29], NewsQA [38], and Quoref [10], etc. However, the reference-based techniques suffer from some major limitations as presented in Section 1. Moreover, each of them tends to target a specific QA format. Also, as presented in the existing work [6], solely testing following the reference-based paradigm on these benchmarks is not extensible and may be biased and insufficient. Different from them, QAQA is an automatic QA testing technique without relying on the manually pre-annotated labels, and guarantees the naturalness of generated test inputs as much as possible through a semantics-guided sentence search process for sentence-level mutation.

Recently, Chen et al. [6] proposed the state-of-the-art automatic QA software testing technique, i.e., QAAskeR, which does not rely on manually pre-annotated labels. Specifically, QAAskeR first synthesizes a pseudo fact based on a question and the answer produced by the QA software under test, and then generates a new question and its expected answer according to the pseudo fact. If the actual answer for the new question is not consistent with the expected answer, a bug in QA software is detected. More details (including the limitations) on QAAskeR have been presented in Section 1. Similar to QAAskeR, QAQA also follows the idea of metamorphic testing to get rid of the dependence on manually pre-annotated labels, but designs totally different metamorphic relations. On the one hand, QAQA designs metamorphic relations by comprehensively considering the question, context, and answer produced by QA software, while QAAskeR ignores the role of the context that is an important component of the input for QA software. On the other hand, compared with QAAskeR, QAQA ensures *high precision* of testing results by designing sentence-level mutation operators in our metamorphic relations and the enhanced answer consistency measurement, as well as the *naturalness* of generated test inputs as much as possible by incorporating the semantics-guided sentence search process. Our experiments have confirmed the superiority of QAQA over QAAskeR in these aspects.

Machine translation system testing. There are some related work on other NLP software testing in the literature [13–15, 28, 35, 36]. The most typical work in this area is machine translator testing. The majority of these techniques also follow the idea of metamorphic testing. For example, Pesu et al. [28] compares the results obtained by directly translating from a source language to a target language with those by indirectly translating through an intermediate language to test machine translators. He et al. [14] proposed *SIT* based on the metamorphic relation where the translation results of similar source sentences should exhibit similar

sentence structures. Gupta et al. [13] proposed *PatInv* whose insight is that sentences with different meanings should not have the same translation results. He et al. [15] proposed *Purity*, which generates reverentially transparent inputs and detects inconsistencies by checking whether they have similar translation results under different contexts. Sun et al. [35] proposed *TransRepair* by comparing the translation results of a sentence and its mutated sentence by replacing a word with another semantically similar word for machine translator testing. They further proposed *CAT* [36] to extend TransRepair by identifying word replacement with controlled impact.

Different from them, QAQA aims to test QA software. Although the two kinds of software are NLP software, they have quite different characteristics (such as different formats of test inputs and different functionalities), and thus these machine translator testing techniques cannot be applied to test QA software. Indeed, QAQA designs totally different metamorphic relations with these metamorphic testing based techniques in machine translator testing.

7 CONCLUSION

This work proposed a novel sentence-level mutation based metamorphic testing approach, QAQA, to precisely detecting bugs in QA software. QAQA uses semantics-guided search to conduct sentence-level mutation, and five metamorphic relations to detect bugs. Our evaluation shows that compared to state-of-the-art, QAQA detects more bugs (8,133 vs. 6,601), has a larger MR violation rate (43.72% vs. 35.48%), and much more precise bug detection (97.67% vs. 49.00%). Semantics-guided search is also demonstrated to yield much more natural test cases than random search (80.67% vs. 4%). In the future, we plan to optimise the automatic test oracle when comparing the similarity between the output answer and the ground truth answer, to further reduce the issue of false positive bugs.

ACKNOWLEDGMENTS

We thank all the anonymous reviewers for their valuable comments on this work. This work was supported by the National Natural Science Foundation of China under Grant Nos. 62002256 and U1836214.

REFERENCES

[1] Accessed: 2022. AlphaChat. https://www.alphachat.ai/.
[2] Accessed: 2022. NeuralCoref. https://github.com/huggingface/neuralcoref/.
[3] Accessed: 2022. Siri. https://www.apple.com/siri/.
[4] Junjie Chen, Yihua Liang, Qingchao Shen, and Jiajun Jiang. 2022. Toward Understanding Deep Learning Framework Bugs. *arXiv preprint arXiv:2203.04026* (2022).
[5] Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. 2020. Practical accuracy estimation for efficient deep neural network testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 4 (2020), 1–35.
[6] Songqiang Chen, Shuo Jin, and Xiaoyuan Xie. 2021. Testing Your Question Answering Software via Asking Recursively. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 104–116.
[7] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 2020. Metamorphic testing: a new approach for generating next test cases. *arXiv preprint arXiv:2002.12543* (2020).
[8] Christopher Clark and Matt Gardner. 2017. Simple and effective multi-paragraph reading comprehension. *arXiv preprint arXiv:1710.10723* (2017).
[9] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044* (2019).
[10] Pradeep Dasigi, Nelson F Liu, Ana Marasović, Noah A Smith, and Matt Gardner. 2019. Quoref: A reading comprehension dataset with questions requiring coreferential reasoning. *arXiv preprint arXiv:1908.05803* (2019).
[11] Steffen Eger and Yannik Benz. 2020. From Hero to Z\'eroe: A Benchmark of Low-Level Adversarial Attacks. *arXiv preprint arXiv:2010.05648* (2020).
[12] Mansi Gupta, Nitish Kulkarni, Raghuveer Chanda, Anirudha Rayasam, and Zachary C Lipton. 2019. Amazonqa: A review-based question answering task. *arXiv preprint arXiv:1908.04364* (2019).
[13] Shashij Gupta, Pinjia He, Clara Meister, and Zhendong Su. 2020. Machine translation testing via pathological invariance. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 863–875.
[14] Pinjia He, Clara Meister, and Zhendong Su. 2020. Structure-invariant testing for machine translation. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*. IEEE, 961–973.
[15] Pinjia He, Clara Meister, and Zhendong Su. 2021. Testing machine translation via referential transparency. In *2021 IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 410–422.
[16] Lynette Hirschman and Robert Gaizauskas. 2001. Natural language question answering: the view from here. *natural language engineering* 7, 4 (2001), 275–300.
[17] Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328* (2017).
[18] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics* 8 (2020), 64–77.
[19] Hidetaka Kamigaito and Manabu Okumura. 2020. Syntactically Look-Ahead Attention Network for Sentence Compression. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 05 (2020), 8050–8057. https://doi.org/10.1609/aaai.v34i05.6315
[20] Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 252–262.
[21] Daniel Khashabi, Tushar Khot, and Ashish Sabharwal. 2020. More bang for your buck: Natural perturbation for robust question answering. *arXiv preprint arXiv:2004.04849* (2020).
[22] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. Unifiedqa: Crossing format boundaries with a single qa system. *arXiv preprint arXiv:2005.00700* (2020).
[23] Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics* 6 (2018), 317–328.
[24] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics* 7 (2019), 453–466.
[25] Shayne Longpre, Kartik Perisetla, Anthony Chen, Nikhil Ramesh, Chris DuBois, and Sameer Singh. 2021. Entity-based knowledge conflicts in question answering. *arXiv preprint arXiv:2109.05052* (2021).
[26] Adam Lopez. 2008. Statistical machine translation. *Comput. Surveys* 40, 3 (2008), 1–49.
[27] Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2019. Multi-hop reading comprehension through question decomposition and rescoring. *arXiv preprint arXiv:1906.02916* (2019).
[28] Daniel Pesu, Zhi Quan Zhou, Jingfeng Zhen, and Dave Towey. 2018. A Monte Carlo Method for Metamorphic Testing of Machine Translation Services. In *2018 IEEE/ACM 3rd International Workshop on Metamorphic Testing*. 38–45.
[29] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. *arXiv preprint arXiv:1806.03822* (2018).
[30] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).
[31] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).
[32] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. *arXiv preprint arXiv:2005.04118* (2020). https://doi.org/10.18653/v1/2020.acl-main.442
[33] Patrick Schramowski, Cigdem Turan, Nico Andersen, Constantin Rothkopf, and Kristian Kersting. 2021. Language models have a moral dimension. *arXiv preprint arXiv:2103.11790* (2021).
[34] Qingchao Shen, Haoyang Ma, Junjie Chen, Yongqiang Tian, Shing-Chi Cheung, and Xiang Chen. 2021. A comprehensive study of deep learning compiler bugs. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 968–980.

[35] Zeyu Sun, Jie M. Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic Testing and Improvement of Machine Translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 974–985. https://doi.org/10.1145/3377811.3380420

[36] Zeyu Sun, Jie M Zhang, Yingfei Xiong, Mark Harman, Mike Papadakis, and Lu Zhang. 2022. Improving machine translation systems via isotopic replacement. In *Proceedings of the 2022 International Conference on Software Engineering*.

[37] Alon Talmor and Jonathan Berant. 2019. MultiQA: An empirical investigation of generalization and transfer in reading comprehension. *arXiv preprint arXiv:1905.13453* (2019).

[38] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A Machine Comprehension Dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*. 191–200. https://doi.org/10.18653/v1/w17-2623

[39] Shufan Wang, Laure Thompson, and Mohit Iyyer. 2021. Phrase-bert: Improved phrase embeddings from bert with an application to corpus exploration. *arXiv preprint arXiv:2109.06304* (2021).

[40] Zan Wang, Ming Yan, Junjie Chen, Shuang Liu, and Dongdi Zhang. 2020. Deep learning library testing via effective model generation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 788–799.

[41] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing test inputs for deep neural networks via mutation analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 397–409.

[42] Caiming Xiong, Victor Zhong, and Richard Socher. 2016. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604* (2016).

[43] Ming Yan, Junjie Chen, Xiangyu Zhang, Lin Tan, Gan Wang, and Zan Wang. 2021. Exposing numerical bugs in deep learning via gradient back-propagation. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 627–638.

[44] Zhuosheng Zhang, Hai Zhao, and Rui Wang. 2020. Machine reading comprehension: The role of contextualized language models and beyond. *arXiv preprint arXiv:2005.06249* (2020).

[45] Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. 2021. Retrieving and reading: A comprehensive survey on open-domain question answering. *arXiv preprint arXiv:2101.00774* (2021).