Testing Your Question Answering Software via Asking Recursively

Songqiang Chen*†, Shuo Jin*†, Xiaoyuan Xie*†‡
*School of Computer Science, Wuhan University, China
i9schen@gmail.com, imjinshuo@whu.edu.cn, xxie@whu.edu.cn

2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 978-1-6654-0337-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ASE51524.2021.9678670

***Abstract*—Question Answering (QA) is an attractive and challenging area in NLP community. There are diverse algorithms being proposed and various benchmark datasets with different topics and task formats being constructed. QA software has also been widely used in daily human life now. However, current QA software is mainly tested in a reference-based paradigm, in which the expected outputs (labels) of test cases need to be annotated with much human effort before testing. As a result, neither the just-in-time test during usage nor the extensible test on massive unlabeled real-life data is feasible, which keeps the current testing of QA software from being flexible and sufficient. In this paper, we propose a method, QAASKER, with three novel Metamorphic Relations for testing QA software. QAASKER does not require the annotated labels but tests QA software by checking its behaviors on multiple recursively asked questions that are related to the same knowledge. Experimental results show that QAASKER can reveal violations at over 80% of valid cases without using any pre-annotated labels. Diverse answering issues, especially the limited generalization on question types across datasets, are revealed on a state-of-the-art QA algorithm.**

***Index Terms*—question answering, testing and validation, recursive metamorphic testing, natural language processing**

I. Introduction

With the booming development of Natural Language Processing (NLP) techniques, machine has been able to process many tasks. Among them, Question Answering (QA) is one challenging but attractive goal that requires machine to understand the human language and infer information from it as the human do [1], [2]. Given a question, QA software intelligently comprehends the relevant information from a lengthy reference passage or one huge knowledge base and returns the deduced answer. QA software has been widely used in daily human life now. For example, many intelligent devices are equipped with a virtual assistant, such as Siri from Apple [3] and DuerOS from Baidu [4], which can provide the QA service.

Recently, we have seen many algorithms being proposed to improve the performance of QA software. Meanwhile, various benchmark datasets with distinct topics and task formats have been constructed to evaluate how well machine can answer the questions [5]–[8]. **Nevertheless, the testing methods for QA software are still primitive and thin.** Specifically, current QA testing practices mainly adopt **the reference-based paradigm**. And when performing a reference-based test, the researchers or engineers have to **first manually annotate the labels** (correct answers) for the test cases (assigned questions), which requires much human effort [6], [9]. Afterwards, QA software is tested by comparing its outputs with the annotated labels. As a result, these testing practices of QA software are mandatorily relying on the existing well-annotated datasets.

However, the reference-based test paradigm has some limitations due to its reliance on the pre-annotated labels. **First, it cannot support the "just-in-time test" for QA software**, which requires an immediate issue detection on the returned answers to unlabeled questions. But such a kind of testing is actually inevitable and necessary in the daily usage. Let us consider the common usage scenario of QA software, where the user inputs one question to which she is looking for the answer. After getting an answer from QA software, she needs to make a quick decision on whether to trust this answer or not, without any pre-annotated labels. This process could be seen as one test execution followed by an immediate issue detection on which the decision is based, but the current reference-based test paradigm is obviously not designed to support such a process. And for the real-life usage, it is also very common to see the users directly trust the QA software to have passed this test and given a reliable answer, because they have barely any clues on the correctness of these answers. However, since the reliability of QA software is not always guaranteed because of the complexity and intractability of the neural networks, it could be very risky to trust the outputs without any inspection. **Secondly**, the reference-based test can only be performed on the existing well-annotated benchmarks, which **may confine the test sufficiency on QA software and hinder the understanding of its real performance**. Actually, recent studies found that some existing benchmarks introduce bias of topics and task formats because of their limited size and diversity [10], [11]. As a result, some vital functionalities of QA software may not have been well tested yet. However, the only way to perform more reference-based tests with the massive and continuously increasing unlabeled real-life data is to pay a lot of human effort to annotate them all the time. This is evidently inefficient and infeasible. In addition, **over-reliance on the manually annotated labels could also hurt the accuracy of test results**, because it has been revealed that some of the manually annotated labels that are regarded as the "golden references" in benchmarks could be erroneous [12].

All in all, it is of great necessity to provide a new testing method of QA software that does not need any pre-annotated labels, so that a just-in-time test during the usage with efficient and effective issue detection becomes feasible, massive unla-

†Equal contribution and co-first authors.
‡Corresponding author.

978-1-6654-0337-5/21/$31.00 ©2021 IEEE
DOI 10.1109/ASE51524.2021.00020

beled data can be potentially leveraged to conduct the more sufficient tests for QA software, and errors in labels will no longer bring any impact on the test.

In this paper, we propose a method named QAASKER and design three novel Metamorphic Relations (MRs) to achieve this goal. Specifically, QAASKER tests QA Software **by asking recursively**. Given a test input, it first synthesizes a pseudo fact according to the question from this (source) input and the corresponding answer (source output) from QA software. After that, it raises one new question based on the pseudo fact and **recursively asks** QA software for the new answer (follow-up output). Finally, it checks the new answer against the pseudo fact. If both the source and follow-up outputs are correct, the pseudo fact should hold and the new answer would conform with the expected answer derived from it. Otherwise, at least one of the source output and the follow-up output is wrong.

We have performed comprehensive experiments to evaluate the effectiveness of QAASKER. Specifically, we first use it to test a state-of-the-art QA algorithm, UnifiedQA [13], with the test inputs from three classic benchmarks, i.e., SQuAD2 [5], BoolQ [6], and NatQA [7]. The results show that QAASKER can reveal many violations on UnifiedQA at over 80% of valid cases without referring to any annotated labels. The revealed answering issues include the missing answers, nonrecognition of question types, etc. The potential generalization issue of UnifiedQA on question types has also been revealed. Besides, we found that expanding the training data with the proposed MRs could help to fix the revealed defects. We also preliminarily explore the usefulness of QAASKER on a popular real-life QA application, the Google Search service [14].

In summary, this work makes the following contributions:

- We propose a method named **QAASKER** to test **Q**uestion **A**nswering software via **Ask**ing **R**ecursively. It gets rid of the dependency on the manually annotated ground truth labels of test cases and therefore enables both the flexible just-in-time test during usage and the extensible test with massive real-life unlabeled data for QA software.
- We design and implement three novel Metamorphic Relations in QAASKER. They check the consistency among the input question and output answer pairs related to the same pseudo fact, where the input questions are of distinct types (e.g., general questions and wh-questions) or asking for different objects in the pseudo fact.
- We perform comprehensive experiments to evaluate the effectiveness of QAASKER. Results show that QAASKER can successfully reveal many violations at over 80% of valid cases without referring to any annotated labels on the model built with a state-of-the-art QA algorithm. We also show several types of the revealed answering issues, especially the limited generalizability on question types across datasets.
- We demonstrate and discuss the usage of QAASKER on the real-life QA software by taking an initial sip on the Google Search service.

The tool, replication package, and specific implementation details for this paper are available online at [15].

The rest of this paper is structured as follows. Section II introduces the background and motivation of this work. Section III elaborates the details of QAASKER, with three novel MRs proposed. Afterwards, Section IV and Section V describe the settings and the results of the evaluation on QAASKER, respectively. Next, Section VI discusses the real-life usage of QAASKER. Section VII presents the threats to validity and Section VIII lists the related works. Finally, Section IX draws a conclusion and lists our future work.

II. PRELIMINARIES

A. Question Answering Software

Question Answering (QA) has been a hot research topic for a long time. It is omnipresent in various domains in our daily life, such as virtual assistants [1], E-commerce services [16], and healthcare [17], [18]. QA tasks are generally divided into *closed-world* and *open-world* [16]. The closed-world QA is to answer each given question **based on the attached reference passage**. And the open-world QA is a new emerging task that demands questions as the only input. It usually requires to first retrieve the reference from the web or construct a knowledge base before performing comprehension and inference [19].

In this paper, we mainly target the closed-world QA software, considering many relevant algorithms and datasets are available. In fact, the closed-world QA is still playing the core role in the open-world QA systems [19]. Therefore, our method can also work on the open-world QA software and we briefly discuss such usages in Section VI as well.

Closed-world QA tasks can be solved with various methods. Sometimes, we can transfer the pre-trained language models to these tasks without much effort. For example, simply fine-tuning ROBERTa [20] and T5 [21] can just lead to promising performance on several closed-world QA task formats, such as the span extraction and the boolean judgment [22]. There are also many specially-designed neural networks proposed to improve question answering by imitating human reading skills [23]–[25]. Recently, Khashabi et al. [13] unify the views of the closed-world QA task formats into one framework to build the general format-agnostic QA systems. They propose a method named UnifiedQA, which can solve the closed-world QA tasks in distinct formats with one single model and has shown pretty promising performance on par with or better than the format-agnostic models on many benchmark datasets.

In the meantime, NLP researchers have constructed a few benchmark datasets and some leaderboards to study the performance and promote the improvement of the QA algorithms [2], [26], [27]. On one hand, they design the datasets of many task formats, such as the span extraction [5], [8], [28], the boolean judgment [6], [29], and the free-form answering [7], [9]. On the other hand, they build corpora in many domains, including customer service [16], clinical and biomedical [17], [18], real-life search queries [6], [7], etc.

B. Motivation

As introduced above, QA software has been widely used in daily human life, thus there is an urgent demand to assure

TABLE I
A MOTIVATING REAL-LIFE EXAMPLE

Reference Passage ***(The News)***	... the 13th season of "SuperNatural" ... premiered on October 12, 2017 on the CW and concluded on May ... the 7th season of the American TV series "Arrow" ... premiered on the CW on October 15, 2018 and is set ...
Question 1 **Answer 1**	When does the new episode of Arrow come out? October 12
Question 2 **Answer 2**	Whose new episode comes out on October 12? SuperNatural

the quality of its returned answers and reveal its undisclosed defects. Currently, almost all the NLP models, including the core models in QA software, are mainly tested in the reference-based paradigm [11], [30]. As explained in Section I, in this test paradigm, the researchers or engineers must first obtain a well-annotated benchmark dataset, which means the existing benchmarks are mandatory during testing QA software. Once presented with the output answers to unannotated questions, current testing methods cannot automatically make a decision about whether there is any problem in the output answers.

In fact, such a decision is inevitable, and it is of great necessity to support it. Let us first consider a real-life example[1]. Supposing that Tom is a super fan of the American superhero TV series, he is looking forward to the 7th season of *Arrow*. One day he receives a piece of news (as shown in Table I) telling that *Arrow* will succeed *SuperNature* that is currently on show. To avoid reading the lengthy news, he asks a virtual assistant, which is built upon one UnifiedQA model, to read the news and answer "*When does the new episode of Arrow come out?*" (Question 1). After few seconds, he receives an answer "*October 12*" (Answer 1). Without having enough time to carefully verify it with the lengthy news, Tom chooses to trust this answer. However, when the day comes, Tom turns on the TV with great expectation but only finds that *Arrow* does not come out, which largely dampens his enthusiasm. This does bring an unpleasant experience. But if we consider Tom's question a test case, it is indeed not easy to automatically and quickly verify the output answer now, because no ground truth label is available. Thus, **to avoid such unhappiness and even more serious issues in other critical application domains**, a new testing method that does not require the label is desired to support the immediate issue detection on the returned answers to the unlabeled questions for QA software.

And such a method is also **necessary for the more comprehensive tests for QA software, even if there exist a few benchmark datasets**. As mentioned in Section I, the benchmarks are found to be imperfect. Specifically, many existing benchmark datasets are found to have the bias of topics and task formats and therefore may hinder the understanding of real-life performance [10], [11]. As a result, it is far from being sufficient to solely rely on the existing finite benchmarks to test QA software. Meanwhile, it can also be very expensive to construct new well-annotated benchmarks because it requires much human effort to annotate the correct answers for the new questions in them [6], [9]. In addition, the manual annotation could also introduce some errors [12], thus hurts the accuracy of the test results.

Therefore, in this work, we aim to propose a testing method to achieve these goals. The proposed method should not rely on the annotated ground truth labels, thus it can provide a just-in-time test with efficient and effective issue detection and leverage massive unlabeled data to perform the extensible and abundant tests for QA software.

C. Metamorphic Testing

Metamorphic Testing (MT) is a proper candidate solution to bypass the labels of test cases, as it was proposed to reuse the passed test cases and alleviate the oracle problem during software testing [31], [32]. MT does not require any inspection on the correctness of each individual output. Instead, it checks whether multiple outputs satisfy the specified relations, namely the Metamorphic Relations (MRs). One famous object of MT is sin function. Verifying the correctness of $sin(x)$ given an arbitrary x is very expensive. In order words, we encounter the oracle problem when testing sin function. But checking the relation of $sin(x) = -sin(-x)$ is straightforward. In this example, $sin(x) = -sin(-x)$ is called the Metamorphic Relation (MR), which can be also rephrased as: if x (the source test input) is negated to $-x$ (the follow-up test input), their outputs are also opposite to each other. MT has been used to test various software and systems, such as the supervised classifiers [33] and the unsupervised clusters [34]. And Zhou et al. [35] use MT to perform a system/service level validation for the search engines, where they mainly construct the follow-up inputs by considering the information in the source outputs as additional query restrictions. Recently, we have also seen MT being widely used for testing many deep learning applications, such as autonomous driving systems [36]–[39] and language translation services [40]–[44].

III. METHODOLOGY

A. A Recursive Metamorphic Testing Method for QA Software

Let us revisit the example in Section II-B. Supposing that Jack is another super fan of *Arrow*, he asks the same question as Tom does and gets a same answer, that is, "*October 12*", from his virtual assistant. He does not have time to carefully verify this output answer as well. But unlike Tom, by seeing this answer, Jack decides to go further and ask the assistant a new question, "*Whose new episode comes out on October 12?*" (Question 2). As shown in Table I, at this time the assistant replies "*SuperNatural*" (Answer 2). By comparing these two answers, Jack is then confused, since "*Arrow*" is not included in the second answer as he has expected. But the good thing is: even if Jack may not be clear about which answer is wrong, he has already had some clues that this QA virtual assistant is not reliable and at least one of the two answers is incorrect.

This example illustrates the basic idea of this paper. To break the reliance on the annotated labels during testing, we

[1]This example is based on the real output of our trained UnifiedQA [13] model for an actual test case from the NatQA [7] dataset.

propose a method, **QAASKER**, to test **QA** software by **Ask**ing **R**ecursively. The input of QAASKER is the QA software under test (SUT) and a list of unlabeled closed-world queries (each of which contains a reference passage and a question, but no manually labeled answers for the questions are required). The output of QAASKER is a list of the revealed suspicious issues that the tested QA software has made.

By leveraging Metamorphic Testing (MT), QAASKER tests the SUT via checking whether its multiple outputs violate the expected Metamorphic Relations (MRs) instead of comparing each individual output with its ground truth label. The basic idea of the MRs we design in this paper is that **a correct answer should imply a piece of knowledge that always holds**. Specifically, given a test input $s = (p, q)$ (also considered as "source input") where p is the reference passage and q is the question, let us denote the answer given by SUT as a. Then, a piece of knowledge k can be synthesized from q and a. This k is also referred as a "*pseudo fact*". Taking Question 1 and Answer 1 in Table I as an example, by synthesizing Question 1 and Answer 1, we can then obtain a piece of knowledge "*The new episode of Arrow comes out on October 12.*". Obviously, if the answer a is correct, the knowledge k is then a true fact that should always hold. **Based on this synthesized knowledge** k, we next **recursively raise a new question** q'. Let us denote the new answer that SUT returns as a'. If a' is also correct, it should conform with the above knowledge k, regarding q'. Otherwise, at least one of a and a' is erroneous and a violation, as well as an answering issue, is revealed by this pair of input and output. In the above example, we construct Question 2 "*Whose new episode comes out on October 12?*" based on k. Obviously, the knowledge from Answer 2 "*SuperNatural*" and Question 2 does not conform with k, and one issue is therefore revealed. This process does not require the label of s but automatically creates oracles. As a consequence, **it effectively breaks the dependency on the manually annotated labels during testing** and thus provides a possible solution to testing QA software on unlabeled data.

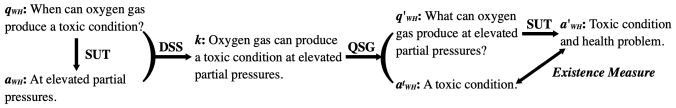
Based on this idea, we propose three novel MRs by considering the consistency among the input question and output answer pairs related to the same knowledge, where the questions are of different types (i.e., the general questions, alternative questions, and wh-questions) or asking for different objects in the knowledge. QAASKER realizes these MRs with three modules, namely the synthesis of knowledge declaration from the given question and answer, the generation of the question from the given knowledge, and the violation measurement on the source and follow-up cases. In the following, we will elaborate the design of the MRs and the modules in detail.

B. Proposed Metamorphic Relations

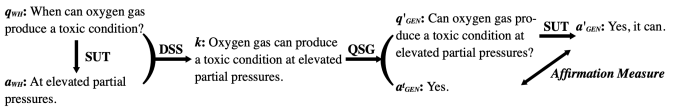
As mentioned above, we propose three MRs to test the QA software by checking its behaviors on multiple questions that are related to a same knowledge. In this section, we introduce the overall idea of each MR. To simplify the demonstration, we denote the question in source and follow-up input as q_{TYPE} and q'_{TYPE}, respectively. The source and follow-up outputs of SUT are denoted as a_{TYPE} and a'_{TYPE} in the same way. The value of *TYPE* is one from {*WH*, *GEN*, *ALT*} as explained in Table II. As a reminder, the reference passage in the follow-up input is the same as the one in the source input.

TABLE II
DEFINITION OF QUESTION TYPES

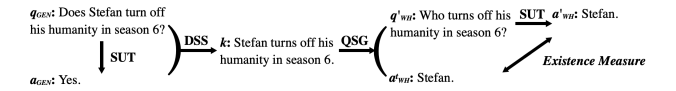
Abbr.	Type	Examples
WH	wh-question	Q: *Who was Emma's brother?* A: *Duke Richard II.* Q: *How many soldiers were in each Tumen?* A: *10,000.*
GEN	general question	Q: *Is this the last year for once upon a time?* A: *Yes.* Q: *Does a cow have to be pregnant to lactate?* A: *No.*
ALT	alternative question	Q: *Is the UK a state or a country?* A: *A country.* Q: *Is a potato a tuber or a vegetable?* A: *A tuber.*



(a) Test Process of MR1



(b) Test Process of MR2



(c) Test Process of MR3

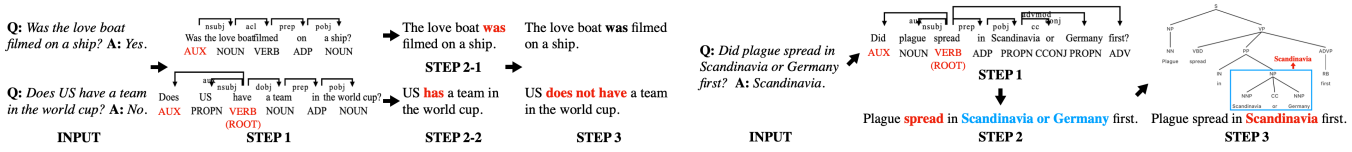
Fig. 1. Proposed Recursive Metamorphic Relations

MR1: Answering new follow-up wh-question derived from the answer of the existing source wh-question.

This MR is eligible for the test inputs with a wh-question on which SUT's output is not "<*NoAnswer*>". As shown in Fig. 1(a), given a wh-question q_{WH}, we first obtain a_{WH} from SUT. Then, a declarative sentence k (i.e., the knowledge) is synthesized from q_{WH} and a_{WH} with the declarative sentence synthesis (DSS) module. After that, we leverage the question sentence generation (QSG) module to generate the new wh-question and the corresponding target answer based on k. As there may be more than one wh-questions available for k, we randomly pick one from them as q'_{WH} and its target answer is denoted as a^t_{WH}. Next, we run SUT with q'_{WH} to obtain a'_{WH}. If the SUT is correct, both a_{WH} and a'_{WH} should be correct, and the knowledge k is a true fact. Since a^t_{WH} is deduced from k, we expect a^t_{WH} is part of a'_{WH}. If a^t_{WH} does not exist in a'_{WH}, at least one of a_{WH} and a'_{WH} is wrong. (More details about the output checking will be introduced in Section III-E1.)

MR2: Answering new follow-up general question derived from the answer of the existing source wh-question.

This MR is also eligible for the test cases whose question is one wh-question with a non-"<*NoAnswer*>" SUT output. Fig. 1(b) shows the overall process of this MR. Similar to the operation in MR1, we first synthesize the declarative sentence k from q_{WH} and a_{WH} with DSS. Next, we use QSG to generate a new general question q'_{GEN} and the corresponding expected target answer a^t_{GEN}, based on k. It is not difficult to find out



(a) Declarative Sentence Synthesis from General Question

(b) Declarative Sentence Synthesis from Alternative Question

Rule	Example
WH be noun$_1$? $\rightarrow$ *noun*$_1$ *be* a_{WH}.	*How is the speed of light in all reference frames? + The same.* $\rightarrow$ *The speed of light is the same in all reference frames.*
WH do noun$_1$ *verb*$_1$ ...? $\rightarrow$ *noun*$_1$ *verb*$'_1$ a_{WH} ...	*What does the sea monster with a female upper body hold in its claws? + A sword.* $\rightarrow$ *The sea monster with a female upper body holds a sword in its claws.*
WH modal noun$_1$ *verb*$_1$ ...? $\rightarrow$ *noun*$_1$ *modal verb*$_1$ a_{WH} ...	*When can oxygen gas produce a toxic condition? + At elevated partial pressures.* $\rightarrow$ *Oxygen gas can produce a toxic condition at elevated partial pressures.*
Whose *noun*$_1$ *be noun*$_2$? $\rightarrow$ a_{WH} *noun*$_1$ *be noun*$_2$.	*Whose theory was the theory of continental drift? + Alfred Wegener.* $\rightarrow$ *Alfred Wegener's theory was the theory of continental drift.*

WH: wh-words like "what", "how", "when", "who", etc. ***modal***: modal words like "can", "must", 'would", etc. ***verb/verb'***: verb phrase and its adaption to the tense and number of auxiliary.

(c) Typical Heuristic Rules for Declarative Sentence Synthesis from Wh-Question

Fig. 2. Process and Example of Declarative Sentence Synthesis

that the a^t_{GEN} should be "Yes". We then run the SUT with q'_{GEN} to obtain a'_{GEN}. If the SUT is correct, both a_{WH} and a'_{GEN} should be correct, and k should be a true fact accordingly. As a result, a'_{GEN} should be consistent with a^t_{GEN}, that is, "Yes" (or other sentences that express an affirmation). Otherwise, an issue is found because there must be at least one error in a_{WH} and a'_{GEN}. (More details about the output checking will be introduced in Section III-E2.)

MR3: Answering new follow-up wh-question derived from the answer of the existing source general or alternative question.

This MR is eligible for the test inputs that have a general question or an alternative question. As shown in Fig. 1(c), we first use DSS to transform the given q_{GEN} (or q_{ALT}) into its declarative form k according to a_{GEN} (or a_{ALT}). After that, as we operate in MR1, we obtain a new wh-question q'_{WH} as well as its expected target answer a^t_{WH} based on k, and run SUT with q'_{WH} to obtain a'_{WH}. If the SUT is correct, then both a_{GEN} (or a_{ALT}) and a'_{WH} should be correct, and the k is a true fact accordingly. As a consequence, a^t_{WH} should exist in a'_{WH}. The absence of a^t_{WH} in a'_{WH} would indicate that at least one in a_{GEN} (or a_{ALT}) and a'_{WH} is erroneous. (More details about the output checking will be introduced in Section III-E1.)

C. Declarative Sentence Synthesis

In this section, we introduce the methods to synthesize the declarative sentence (the knowledge k) from a pair of question and SUT's corresponding output answer. The question could be one of three types, namely general questions, alternative questions, and wh-questions.

1) Declarative Sentence Synthesis from General Question: For a general question q_{GEN} and SUT's answer a_{GEN}, three steps are needed to synthesize the corresponding declarative sentence k. Fig. 2(a) shows this process. Specifically, we first use spaCy toolkit [45] to analyze the token dependency and locate the auxiliary (AUX)[2] in q_{GEN} (step 1). When AUX is a form of *be* or a modal auxiliary, we then move it to the location before the predictive verb (VERB(ROOT)) of the sentence (step 2-1). If AUX is a form of *do*, we remove AUX and transform VERB(ROOT) into the tense and number of AUX with Pattern Library [47] (step 2-2). After that, the declarative sentence k corresponding to q_{GEN} is prepared. Finally, if a_{GEN} is not an affirmation (e.g., "No"), k is further negated (step 3). The final k is returned as the declarative sentence for q_{GEN} and a_{GEN}.

2) Declarative Sentence Synthesis from Alternative Question: It also involves three steps to synthesize the declarative sentence from the given general question q_{ALT} and SUT's answer a_{ALT}. The whole process is shown in Fig. 2(b). The first two steps are similar to the operations in Section III-C1. The difference is that the obtained k after step 2 still contains the alternatives (text in blue). So we adopt the Berkeley Neural Parser [48] to parse the syntax tree of k and then use a_{ALT} to replace the sub-tree rooted at the parent node of "*or*" (step 3). Finally, the obtained k is returned as the declarative sentence for q_{ALT} and a_{ALT}.

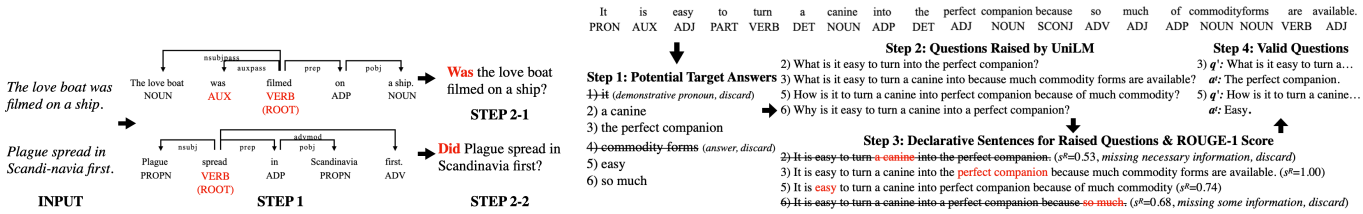
3) Declarative Sentence Synthesis from Wh-Question: To obtain a fairly reliable declarative sentence from the given wh-question q_{WH} and SUT's answer a_{WH}, we design numerous heuristic rules to process the distinct forms of q_{WH}. Due to the limited space in this paper, we only list four basic operations in Fig. 2(c). The detailed rules (e.g., to adapt preposition) could be found in the supplementary material [15]. These operations are also performed based on the token dependency and the Part-of-Speech Tags analyzed with spaCy toolkit [45] on q_{WH}.

D. Follow-up Question Sentence Generation

In this section, we introduce the methods to generate follow-up question sentences from the declarative sentences synthesized by the above module. Two types of questions, namely general questions and wh-questions, could be generated.

1) General Question Sentence Generation: The generation of general questions could be seen as the opposite process to Section III-C1. As shown in Fig. 3(a), given a declarative sentence k, we first locate the predictive verb (VERB(ROOT)) in k

[2]AUX is a non-main verb of the clause, including a modal auxiliary and a form of *be*, *do* or *have* in a periphrastic tense. Details could be found at [46].



(a) General Question Sentence Generation

(b) Wh-Question Sentence Generation

Fig. 3. Process and Example of Follow-up Question Sentence Generation

(step 1). Then, we check if there is an auxiliary (AUX) before VERB(ROOT). If it is, we move the AUX to the beginning of the whole sentence and then the corresponding general question q'_{GEN} is generated (step 2-1). Otherwise, we use Pattern Library [47] to recognize the tense and number of VERB(ROOT) and insert a *do* with suitable tense and number to the beginning of the sentence. The q'_{GEN} is then obtained (step 2-2).

2) Wh-Question Sentence Generation: Generating wh-questions based on the given knowledge is fairly complicated and challenging. It generally consists of two major steps, i.e., choosing proper target answers and producing the corresponding questions. Fig. 3(b) gives an example of this process.

To choose proper target answers from the given declarative sentence k, we first extract noun phrases and adjective phrases from k because they are usually used as the answers of QA software according to our observation on benchmark datasets (step 1). This process is performed based on the Part-of-Speech Tag of each token in k, which is labeled with spaCy toolkit [45]. As a reminder, some unsuitable answers, such as phrases with demonstrative pronouns and a_{WH}, are excluded from $\mathbb{TA}$ (detailed rules could be found at [15]).

With the potential target answers in $\mathbb{TA}$, QAASKER then raises a reasonable question for each of them according to k (step 2). To handle various expression phenomena, we turn to a DL-based end-to-end Language Model, UniLM [49], instead of designing complicated heuristic rules. UniLM has shown fairly promising performance of question generation on SQuAD1 dataset [8]. Specifically, we load the UniLM question generation model that is trained on SQuAD1 and released publicly. QAASKER then inputs k together with each answer ta_j in $\mathbb{TA}$ to the trained model and obtains the corresponding new question set $\mathbb{NQ} = \{nq_1, nq_2, \ldots, nq_n\}$.

However, the quality of the questions raised by UniLM is not always guaranteed. For example, the questions generated for the potential target answer 2) and 6) in the example are unreasonable questions. These questions may lead to potential false positive issues since the MRs do not hold when input is invalid. To avoid such situation, we further design a method to sift the fairly clean and reliable questions from $\mathbb{NQ}$ (step 3). Our basic idea is that for each nq_j, if it is a reasonable question for ta_j according to k, the declarative sentence k'_j created with nq_j and ta_j should be very similar to k. Therefore, we adopt a widely-used sentence similarity metric, ROUGE [50], to provide a similarity score[3] s_j^R between k and each k'_j. If s_j^R is no greater than the pre-defined threshold θ^R, ta_j and nq_j will be erased from $\mathbb{TA}$ and $\mathbb{NQ}$, respectively. According to the result of our preliminary experiments, we set θ^R to be 0.7 in QAASKER. The potential target answer 2) and 6) and their questions in this example are hence filtered out.

Finally, the valid new questions remained in $\mathbb{NQ}$ and their corresponding target answers in $\mathbb{TA}$ will be returned as the candidate new wh-questions for MR1 and MR3 (step 4).

TABLE III
EXAMPLE OF EXISTENCE MEASUREMENT

	egyptian	president	*(maximum)*
president	0.2363	1.0000	1.0000
egypt	0.7443	0.2128	0.7443

E. Violation Measurement

In this section, we introduce the methods designed to measure whether SUT violates the MRs on the given test case. As mentioned in Section III-B, we need to measure if a'_{WH} contains a^t_{WH} (MR1 and MR3) and if a'_{GEN} expresses the affirmation (MR2). QAASKER achieves the measurement by considering the sentence semantic similarity. Specifically, we use the semantic overlap between a'_{WH} and a^t_{WH} to indicate the existence of a^t_{WH} in a'_{WH}, and the affirmation is measured with the semantic similarity between a'_{GEN} and some affirmative expressions like "Yes".

1) Existence Measurement: Whether a^t_{WH} exists in a'_{WH} is measured via checking if there exist words in a'_{WH} sharing semantically similar embedding vectors with every word in a^t_{WH}. Considering the stop words often contain limited semantic information, we do not consider them in this process.

Let us consider an example whose a^t_{WH} is "*the president of egypt*" and a'_{WH} is "*egyptian president*". Table III shows the analysis on this example. Specifically, we first discard the stop words "*the*" and "*of*". Then, for each word in a^t_{WH} (second and third rows), we calculate the *cosine similarity* between it and all the words in a'_{WH} (second and third columns) as suggested in [51]. After that, the maximum similarity for each word in a^t_{WH} is calculated as shown in the right-most column. With this

[3]The similarity score is defined as $s^R(a,b) = \min(R1Precision(a,b), R1Recall(a,b))$, where a and b are two strings while *R1Precision* and *R1Recall* are two sub-metrics in ROUGE-1 score (token-wise ROUGE similarity between two sentences).

method, although "*egypt*" from a^t_{WH} is not in a'_{WH}, a'_{WH} is still considered to contain a^t_{WH} and not a violation, as it contains "*egyptian*" that shares a similar word embedding vector and expresses similar semantic meaning with "*egypt*". Finally, we average all the word-wise maximum similarity into an overall score s^{exi}_{avg} to indicate the existence of a^t_{WH} in a'_{WH}. It will be next compared against a pre-defined threshold θ^{exi}. If s^{exi}_{avg} is no greater than θ^{exi}, a^t_{WH} is considered absent from a'_{WH} and a violation will be reported. We set θ^{exi} to be 0.6000 according to our preliminary experimental results. In this example, s^{exi}_{avg} is calculated as (1.0000+0.7443)/2=0.8722. The SUT is thus considered to pass the test on this test case.

2) Affirmation Measurement: To check whether a'_{GEN} expresses affirmation to the given general question, we propose to calculate the maximum semantic similarity between a'_{GEN} and the word "*Yes*".

TABLE IV
EXAMPLE OF AFFIRMATION MEASUREMENT

	yep	black	*(maximum)*
yes	0.6019	0.2926	0.6019

This process is similar to the measurement of existence. Specifically, as shown in Table IV, a'_{GEN} in this example is "*yep it is black*". We first remove the stop words "*it*" and "*is*". After that, the word-wise cosine similarity is calculated as shown in the second and third columns. Then, we obtain the affirmative score s^{aff} of a'_{GEN}, namely the maximum similarity to "*Yes*". In this example, s^{aff} is 0.6019. There is also a threshold θ^{aff} set as 0.6000 according to our preliminary experimental results. As a result, SUT is considered to pass this test case because of $s^{aff} > \theta^{aff}$.

IV. EXPERIMENTAL SETUP

A. Research Questions

To evaluate QAASKER, we study four research questions:

RQ1: *The overall effectiveness of* QAASKER. In this RQ, we aim to provide a global picture on the effectiveness of QAASKER in revealing the issues of QA software without using the annotated ground truth labels.

RQ2: *Validity of the revealed violations.* Considering the imperfection in most of the NLP generation and measurement methods [49], [50], it is meaningful to understand the factuality of these revealed violations. Therefore, in this RQ, we perform a deeper inspection on these violations to measure their validity.

RQ3: *Types of the revealed true violations.* To provide an intuitive and constructive impression on the revealed issues, in this RQ, we dive into the analysis on the valid violations by locating the erroneous answers (that is, the source or the follow-up outputs), as well as summarizing the types of the answering issues according to their reasons.

RQ4: *Helpfulness to fix the revealed answering issues.* In this RQ, we study the performance of a new model trained on the dataset expanded with the proposed MRs in order to understand if our method is helpful to fix the revealed issues. This is a common paradigm adopted in many works that use MT to test deep learning software [37], [40], [41].

B. Data Preparation

To evaluate the effectiveness of QAASKER on diverse types of questions and tasks, we perform the evaluation experiment on three datasets[4], namely SQuAD2, BoolQ, and NatQA.

- **SQuAD2** is a span extraction dataset, where the answer of each question is a span of words from the reference passage without demanding combination and rephrasing. And when the question is unanswerable, the output is expected to be "*<NoAnswer>*". It contains 140k samples with wh-questions and 2k samples with general or alternative questions in total, which are divided into 130k training samples and 12k test samples.
- **BoolQ** is a dataset totally composed of general questions obtained from Google Search queries and paired with passages from Wikipedia that are considered sufficient to deduce the answer. The answer is expected to be either "Yes" or "No" (or sentences with similar meanings [13]). It has 9.4k training samples and 3.3k test samples.
- **NatQA** is one abstractive QA dataset, which means it requires the model to return answers that are not mere substrings of the reference passage. We use the version provided by UnifiedQA where each question is appended with a reference passage. It includes 98k wh-questions and 299 general and alternative questions[5], which are then divided into 97k training samples and 11k test samples.

For each dataset, we first use the QA software under test to obtain the source outputs of all the test samples. After that, we apply each of the three MRs on its eligible source test samples, respectively. The testing (i.e., violation measurement) is then conducted on the eligible samples.

C. Test Object

In this work, we use QAASKER to test the QA model built with a state-of-the-art QA algorithm, UnifiedQA [13]. As we introduce in Section II-A, UnifiedQA provides a solution to the format-agnostic general QA system by unifying diverse common closed-world QA task formats. It first trains a basic text-to-text model on some seed QA datasets of multiple task formats, during which the reference passages and the questions in natural text are taken as input without using format-specific prefixes. People can then fine-tune this pre-trained model into specialized models for better performance on the specific QA tasks. It is reported to have achieved promising performance upon SQuAD2, BoolQ, and NatQA with some latest language models like T5 [21].

Since UnifiedQA can solve various QA tasks with a unified model, we use all three datasets to train only one UnifiedQA model as the test object. Specifically, we collect the training

[4]We use the pre-processed datasets provided by UnifiedQA [13]. They can be found at https://console.cloud.google.com/storage/browser/unifiedqa/data.

[5]NatQA includes some questions in miscellaneous and informal forms, e.g., "total number of death row inmates in the us?".

samples from SQuAD2, BoolQ, and NatQA to form a hybrid training set with 236,422 samples. As suggested in [13], we use this hybrid training set to fine-tune the pre-trained T5-large-based UnifiedQA model and regard the checkpoint with the highest *Exact Match* (EM) score on the hybrid test set as the final test object. The model is trained with one NVIDIA GeForce RTX3090 GPU with 24GB memory, during which the batch size is 3, the learning rate is 2e-5, the loss accumulates every 5 steps, the EM-based evaluation is conducted per 5000 steps, and the early stop tolerance is 10 times.

V. Results and Analysis

A. RQ1: The Overall Effectiveness of QAASKER

To evaluate the overall effectiveness of QAASKER in revealing the answering issues, we calculate SUT's violation rates (the ratio of the violated test cases in all eligible test cases) at each MR on the three datasets, which are shown in Table V.

TABLE V
VIOLATION RATE AT EACH MR ON THREE DATASETS

Dataset	MR1	MR2	MR3
SQuAD2	37.05%	65.85%	90.91%
BoolQ	–	–	72.78%
NatQA	51.98%	96.92%	46.15%

–: MR1 and MR2 cannot be applied on BoolQ as it only contains general questions.

From the result, we surprisingly found that all MRs have revealed many violations on the datasets. This demonstrates the effectiveness of QAASKER to **reveal the answering issues without the need for the ground truth labels**. Furthermore, we also found that MR2 and MR3, which involve the transformation of question types, have revealed relatively more violations than MR1. This is interesting because UnifiedQA has shown promising ability to solve wh-questions and general questions on SQuAD2 and BoolQ, respectively, according to the reference-based tests [13]. But according to our results, UnifiedQA fails to return proper answers to the general question and wh-question related to the same knowledge on these two datasets. From this point, we conjecture that UnifiedQA might overfit the training samples and therefore could only pass the test cases whose question is of the frequent types among the training samples from their corresponding datasets. This **indicates the potential insufficient generalization of UnifiedQA** to figure out the questions of distinct types across datasets, which is vital in unifying QA solutions [13].

B. RQ2: Validity of the Revealed Violations

By obtaining quite a few violations in RQ1, we are particularly interested in evaluating the validity of the revealed violations. We perform a manual inspection on these violations. Specifically, if there is indeed at least one incorrect answer in the source and follow-up outputs, we call the corresponding violation **"valid"**.

We consider this inspection meaningful, because: (1) Apart from reporting the violation rates, it is also necessary to give a deep understanding on the factuality of the revealed violations. (2) Generation of wh-questions and measurement of semantic similarity remain challenging tasks and cannot be guaranteed to be 100% perfect and precise with current NLP techniques [49], [50]. Therefore, it is necessary to check if the violations are due to the incorrect answers or the imperfection in the sentence generation and similarity measurement.

TABLE VI
VALIDITY RATE OF THE REVEALED VIOLATIONS

Dataset	MR1	MR2	MR3
SQuAD2	81/100 (81%)	100/100 (100%)	9/10 (90%)
BoolQ	–	–	87/100 (87%)
NatQA	85/100 (85%)	100/100 (100%)	5/6 (83%)

A co-author of this paper and another volunteer student participate in the manual inspection. Both of them are proficient in English. They are required to perform the inspection independently, without discussing it with each other. Given an MR and a dataset, if more than 100 violations are revealed, they examine the validity of the randomly picked 100 violations. Otherwise, they check the validity of all the violations.

After the inspection on the randomly picked violations, we perform Cohen's Kappa statistics [52]. The agreement rate between two inspectors is substantial (0.79) and the inspectors discuss and settle the disagreement at last. Thus, we consider the validity rate of violations in this inspection can be referred as a fairly reasonable indicator to the overall validity of our experimental results. If this rate is fairly high, it means that the effectiveness reported in RQ1 is convincing, and most of the revealed violations are meaningful and should be seriously considered by the developers and the users of QA software.

The final result is shown in Table VI, from which we found that the inspected violations revealed by MR2 are all valid, and over 80% of the inspected violations revealed by MR1 and MR3 are valid as well. These rates are considered acceptable when compared with the precision in similar testing methods for the machine translation software [40]–[42]. We also review the false positive violations and found that they mainly result from the limitation of the semantic equivalence measurement and the question generation on few corner cases. For instance, it is not simple to identify the expected answer "yesun temur" as semantically contained in the output answer "yesün temür". And it is also not easy to filter out the questions with minor flaws, such as "Seven episodes are going to be in what season?" for knowledge "Seven episodes are going to be in *game of thrones season 7*" (the target answer is in italic).

To conclude, the high validity rates indicate **the effectiveness of QAASKER is meaningful and convincing**. They also show the certain reliability of the design and implementation in our QAASKER regarding the quality of wh-question generation and semantics similarity measurement.

C. RQ3: Types of the Revealed True Violations

In this RQ, we further study the valid true violations found in RQ2, by locating the erroneous answers (that is, the source or the follow-up answer) as well as summarizing the reasons.

TABLE VII
NUMBER OF ERRONEOUS ANSWERS ON TRUE VIOLATIONS

Dataset	MR1	MR2	MR3
SQuAD2	22 , 59	25 , 75	4 , 5
BoolQ	–	–	18 , 69
NatQA	44 , 41	58 , 42	0 , 5

1. "A , B" means that in A (B) violations the source (follow-up) output is wrong.
2. As a reminder, when the source answer is wrong, the correctness of the follow-up answer cannot be assessed and thus we do not consider it wrong.

We first locate the erroneous answer in the true violations. The statistics result in Table VII shows that QAASKER can find errors on both source test cases and follow-up test cases. Moreover, we found that in the violations revealed by MR2 and MR3 on SQuAD2 and BoolQ, respectively, more issues are blamed for the follow-up answer than the source answer. Since the follow-up questions formulated with MR2 and MR3 are of the fairly unfrequent formats in SQuAD2 and BoolQ, respectively, this phenomenon further supports the conjecture of the limited generalization on question types across datasets, which has been discussed in Section V-A.

Besides, we explore five categories in the true violations to give a systematic and intuitive understanding about the issues revealed by QAASKER. The examples are listed in Table VIII.

<*NoAnswer*> for answerable questions. We first found that UnifiedQA cannot answer some answerable questions. In Example 1, the passage provides obvious evidence to deduce the answer, in which only the word "fund" in the question is replaced with its synonym "meet the cost". However, the UnifiedQA model fails to find the correct answer and merely outputs <*NoAnswer*>.

Format mismatch between the answer and the question. The second major issue is that some answers from UnifiedQA are not in the correct format that corresponds to the assigned questions. For example, the question in Example 2-1 is a wh-question, which desires the concrete name of a film. However, SUT only returns "No". Meanwhile, the answer to the general question in Example 2-2 should be either "Yes" or "No", but SUT wrongly gives an irrelevant verb as the answer.

Irrelevant content of the answer. Although successful in recognizing the type of the question, the UnifiedQA model sometimes gives answers with irrelevant content. Example 3 presents an example of this situation. The model returns an answer "Sky", which is far from the correct answer "Some encrypted broadcasts".

Grammatical error. The UnifiedQA model also returns some answers with grammatical issues, which largely harms the quality of the answers. For instance, in Example 4, an incomplete sentence is given as the answer.

Missing information in the answer. We also notice that the UnifiedQA model may miss some necessary information and give a partially correct answer. Referring to the model's answer in Example 5, though it indicates that Shi Bingzhi is someone's father, the pronoun "his" is ambiguous. It is evidently not an accurate answer yet according to the reference passage.

D. RQ4: Helpfulness to Fix the Revealed Answering Issues

In this RQ, we study the helpfulness of our method in fixing the revealed issues for the SUT. Retraining a model with the samples that are expanded by the proposed MRs is a widely-adopted method to repair the DL models in many works [37], [40], [41]. Inspired by this, we expand the training samples with the three proposed MRs to retrain one new model. The difference of the violation situation between the original and new model would show the helpfulness of the MRs to fix the answering issues.

Specifically, for each sample in the hybrid training set, we use the MRs that are eligible on it to generate a new training sample. To ensure the correctness of the generated samples, we use the ground truth labels of the training samples rather than SUT's output as the answer during the synthesis of declarative sentences. We collect the generated samples together with all the samples in the original hybrid training set to form a new hybrid training set, which includes 537,175 samples. Then, we retrain a new model with the new training set, during which the hyper parameters keep the same as introduced in Section IV-C. The test sets are also kept the same as before.

The test result for the new model is presented in Table IX. We first see that **the improvement is quite substantial**. The violation rates about all MRs on three datasets decrease a lot. Meanwhile, the reference-based test metric, i.e., average EM score, stays fairly stable (original: 0.5574 v.s. new: 0.5483). In addition, the improvement on MR2 and MR3, which involve the transformation of question types, is especially significant. These findings indicate that **the proposed MRs are helpful for improving the performance** of SUT by expanding the existing training samples. These MRs can help the model to correctly handle more questions, notably the ones of the types that are relatively rare in the corresponding dataset.

Meanwhile, we could also find that there still exist many violations. This finding demonstrates that it is not that easy to repair all the issues revealed by QAASKER. Since QAASKER is a testing method per se, from this point we argue that our method is **necessary for the reliability checking of QA software output and the in-depth problem revealing of QA software**. This again confirms the significance of QAASKER **as a testing method.**

VI. DISCUSSION ON REAL-LIFE USAGE

With the development of QA algorithms and other supportive techniques like knowledge graph, some industrial products have been able to provide preliminary QA services. For example, the Google Search service [14] can now return an exact answer or one paragraph with the answer span in bold when we input a wh-question as the query. Thus, we try QAASKER on the Google Search service to take an initial sip of its usefulness on the **real-life QA applications**. And as a search engine, the Google Search service does not require the reference passage as input but retrieves necessary information from web by itself. Therefore, it can also be seen as a representative test object of the **open-world QA software**.

TABLE VIII
EXAMPLES OF REVEALED ANSWERING ISSUES

#	Example 1	Example 2-1	Example 2-2
Reference Passage	... The IPCC receives funding through ... while UNEP meets the cost of the Depute Secretary ...	... the network renewed Carrie Diaries for ... The CW canceled the series after two seasons ...	... Li Tan, the son-in-law of a powerful official, instigated a revolt against Mongol rule in 1262 ...
Question	What does UNEP fund?	What film does not have a season 3?	Did Li Tan lead a revolt in 1262?
Expected Ans	IPCC's deputy secretary	The Carrie Diaries	Yes
UnifiedQA Ans	*<NoAnswer>*	No	Instigated

#	Example 3	Example 4	Example 5
Reference Passage	... Some broadcasts are free-to-air ... some are encrypted and require a monthly subscription ...	... the VideoGuard pay-TV scrambling system owned by NDS, a Cisco Systems company ...	... Shi Tianze was a Han Chinese who lived in the Jin dynasty ... His father was Shi Bingzhi ...
Question	What require to view monthly subscription?	What is Cisco systems?	Who was Shi Bingzhi?
Expected Ans	Some encrypted broadcasts	The parent company of NDS	Shi Tianze's father
UnifiedQA Ans	Sky	The name of the company that	His father

TABLE IX
VIOLATION RATE AFTER FIXING WITH TRAINING DATA EXPANDING

Dataset	MR1	MR2	MR3
SQuAD2	30.62% *(6.43%)*	0.13% *(65.72%)*	48.65% *(42.26%)*
BoolQ	–	–	29.70% *(43.08%)*
NatQA	22.24% *(29.74%)*	0.02% *(96.90%)*	31.58% *(14.57%)*

Values in brackets indicate the improvement to the corresponding rates in Table V.

According to our observation, the Google Search service can mainly answer wh-questions now. Therefore, we only try MR1 on it. Besides, as the returned results vary in forms (e.g., sometimes an exact phrase and occasionally a paragraph with one span in bold), we only perform a small-scale trial by hand as the preliminary exploration. Specifically, we first randomly choose 20 wh-questions from an open-world QA benchmark, MKQA[6] [53]. These 20 wh-questions are used as the source inputs and we manually collect and unify the answers returned from Google Search as the source outputs. After that, we run QAASKER to generate new questions and their target answers based on the source inputs and outputs. We next input the new questions as queries and obtain the search results, from which the follow-up outputs are manually extracted. At last, we perform the violation measurement on all 20 test cases.

As a result, 5 out of 20 test cases trigger a violation[7]. For example, we first query "*When was the first railroad built in the United States?*" and obtain the source answer "*1830*" from Google Search. After that, we query "*In which country was the first railroad built in **1830**?*" whose target answer is "*the United States*". However, Google Search returns an irrelevant answer "*The railroad was first developed in Great Britain...*", which triggers a violation. Actually, the answer to the source input should be "*1827-02-28*" according to the annotated label from MKQA. This demonstrates that QAASKER finds a true erroneous answer returned by the Google Search service. In a word, this trial **shows the potential of QAASKER to reveal the real-life bugs** on daily QA applications. And it also shows the effectiveness of QAASKER on open-world QA software.

[6]All the questions in MKQA can be answered with the public knowledge from the web.

[7]Based on the search results obtained on April 10, 2021.

VII. THREATS TO VALIDITY

The first threat to validity is about the representativeness of the test object and the datasets. Actually, UnifiedQA is a state-of-the-art QA algorithm that has delivered performance on par with or better than the format-agnostic models. Besides, it is the only method to unify the solutions of various QA forms, which approaches the real-life general QA applications [13]. Thus, we consider it as a suitable representative test object in our preliminary exploration. Actually, QAASKER is a black-box testing method that only involves the input and output of QA software. And as we introduce in Section II-A, current QA tasks (and their input and output requirements) can be generally categorized into the closed-world ones and the open-world ones. Therefore, QAASKER should be generalizable to other closed-world and open-world QA software, just in the way in which we test the UnifiedQA model and the Google Search service, respectively. As for the datasets, the adopted benchmarks are all classic and have been widely used in the reference-based testing of QA software and cover the major types of QA tasks [13]. Since we evaluate QAASKER on all of them, we consider the evaluation should have fairly good generalization. We also plan to evaluate QAASKER on more applications and corpora in our future work.

The second threat to validity is about the tools that we use to realize the proposed MRs. As illustrated in Section V-B, the wh-question generation and semantic similarity measurement are not perfect yet because of the limited NLP techniques. To assure the validity of the revealed violations, we have designed various methods to avoid the false positive violations. We also inspected the factuality of the revealed violations. The result shows that over 80% of the inspected violations are valid. This is acceptable when compared to other MT-based test methods for DL software [40]–[42]. And we will keep trying to improve this validity rate in our future work as well.

The last threat to validity comes from the manual inspection and categorization of the revealed violations. To alleviate the bias introduced by the difference of subjective cognition, we delivered a tutorial to the inspectors before the inspection. We have also performed Cohen's Kappa statistics and found the agreement rate between two inspectors is substantial (0.79). And all the disagreements are settled after their discussion.

VIII. Related works

In this section, we discuss the related works in two aspects, i.e., the benchmark datasets proposed for testing QA software and the application of Metamorphic Testing for other Deep Learning software.

A. Benchmark Datasets for QA Software

To test QA systems as well as understand whether machine can intelligently deduce the question as the human do, many works proposed various benchmark datasets [2], [26]. These datasets are with diverse forms of task, including to fill in the blanks [54], [55], judge the correct options [6], [29], [56], extract the relevant spans [5], [8], [28], and return fluent text answers [7], [9]. There are also datasets of the samples with adversarial inputs, such as typos [57] and irrelevant sentences [58], to test the robustness of QA software. But as mentioned in Section I, these datasets may mainly focus on some specific topics and task formats. As a result, solely testing with the reference-based paradigm on these datasets is not extensible and may be biased and insufficient.

Unlike these works, in this paper, we propose a method to test QA software without the demand of annotated labels via asking recursively. It breaks the reliance on the ground truth labels of test cases and hence enables both the flexible just-in-time test and the extensible test that can leverage the massive unlabeled data in real-life usage to test QA software.

B. Metamorphic Testing for Deep Learning Software

To alleviate the oracle problem during testing various Deep Learning (DL) software, quantities of works leverage MT and propose many novel MRs to test the DL models for different tasks.

The Autonomous Driving (AD) systems and the Neural Machine Translation (NMT) services are two typical DL software that attracts many MT-based testing methods. Tian et al. [37] and Zhang et al. [36] propose to test AD against the relation among the steering angles under distinct weather conditions. Zhou et al. [38] combine MT and fuzzing and take the LiDAR point-cloud data of AD into consideration during testing. Wang et al. [39] leverage MT to test the object detection algorithms that are used to build a key component in AD systems. As for the NMT service, researchers propose to check its correctness with MT based on the structure invariance [40], pathological invariance [41], referential transparency [42], etc. In addition to being adopted to test NMT services, MT is also found to be helpful in assessing the quality of input data [43] and repairing the erroneous translations [44] for NMT services.

In this paper, we also adopt MT to test a hot DL application, QA software. Specifically, we propose three novel MRs against the consistency among the input question and output answer pairs that are related to a same knowledge. We also implement three tools, i.e., the declaration synthesis, question generation, and similarity measurement, to realize the proposed MRs.

Besides, we propose to validate the machine reading comprehension (MRC) DL models with MT in our previous work [30]. It aims to provide the MRC models with one systematic and extensible assessment of language understanding capabilities against required linguistic properties. In that work, the follow-up inputs were built on the basis of merely the source inputs and the transformation about several related linguistic properties like synonyms and negations.

Different from that, the QAASKER devised in this work is a recursive metamorphic testing method that constructs follow-up inputs by considering both the source input and the source output. This could involve some more abstract properties, such as the generalizability on question types. And we also evaluate QAASKER on not only the previously used boolean question task, but also the span extraction and the free-form answering tasks. Moreover, we further explore the real-life usefulness of QAASKER on the Google Search service and its helpfulness to repair the revealed issues.

IX. Conclusion and future work

Question Answering (QA) software has been widely used in our daily life. In this paper, we propose a novel recursive Metamorphic Testing method named QAASKER with three Metamorphic Relations. QAASKER tests QA software through checking its behaviors on multiple recursively asked questions that are related to the same knowledge. It breaks the reliance on the pre-annotated labels of test cases, thus enables both the flexible just-in-time test during usage and the extensible test with massive unlabeled data for QA software, which cannot be supported by the current reference-based test paradigm. We evaluate the effectiveness of QAASKER through using it to test a state-of-the-art unified QA algorithm and the Google Search service. The comprehensive experimental results demonstrate that QAASKER is able to reveal quantities of valid violations that depict diverse answering issues.

We have planned plenty of future work directions. First, we would like to further evaluate the effectiveness of QAASKER on more applications and corpora. It would be interesting and significant to see the defects revealed on other QA software, especially the issues about some essential functionalities like generalization and the problems that may have been concealed by the insufficient reference-based tests. In addition, we will also try to design new MRs by considering more properties of QA software and keep improving the validity of the revealed violations. We are pretty interested to explore and strengthen QAASKER on repairing the revealed issues as well.

Acknowledgment

We first sincerely appreciate the positive acknowledgment and the very kind suggestions from the anonymous reviewers. We would also like to thank Yuanxiang Ji for volunteering in the manual inspection practices of our evaluation experiment. This work was partially supported by the National Key R&D Program of China under the grant number 2020AAA0107803, and the National Natural Science Foundation of China under the grant numbers 61972289 and 61832009. And the numerical calculations in this work have been partially done on the supercomputing system in the Supercomputing Center of Wuhan University.

REFERENCES

[1] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng, "MS MARCO: A human generated machine reading comprehension dataset," in *Proceedings of the 2016 Workshop on Cognitive Computation: Integrating neural and symbolic approaches co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016*, ser. CEUR Workshop Proceedings, T. R. Besold, A. Bordes, A. S. d'Avila Garcez, and G. Wayne, Eds., vol. 1773. CEUR-WS.org, 2016.

[2] Z. Zhang, H. Zhao, and R. Wang, "Machine reading comprehension: The role of contextualized language models and beyond," *CoRR*, vol. abs/2005.06249, 2020.

[3] Apple Inc., "Apple siri," https://www.apple.com/siri/.

[4] Baidu Inc., "Baidu dueros," https://dueros.baidu.com/.

[5] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," in *Proceedings of the 2018 Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, I. Gurevych and Y. Miyao, Eds. Association for Computational Linguistics, 2018, pp. 784–789.

[6] C. Clark, K. Lee, M. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "Boolq: Exploring the surprising difficulty of natural yes/no questions," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 2924–2936.

[7] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. P. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, K. Toutanova, L. Jones, M. Kelcey, M. Chang, A. M. Dai, J. Uszkoreit, Q. Le, and S. Petrov, "Natural questions: a benchmark for question answering research," *Trans. Assoc. Comput. Linguistics*, vol. 7, pp. 452–466, 2019.

[8] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, J. Su, X. Carreras, and K. Duh, Eds. The Association for Computational Linguistics, 2016, pp. 2383–2392.

[9] W. He, K. Liu, J. Liu, Y. Lyu, S. Zhao, X. Xiao, Y. Liu, Y. Wang, H. Wu, Q. She, X. Liu, T. Wu, and H. Wang, "Dureader: a chinese machine reading comprehension dataset from real-world applications," in *Proceedings of 2018 the Workshop on Machine Reading for Question Answering@ACL 2018, Melbourne, Australia, July 19, 2018*, E. Choi, M. Seo, D. Chen, R. Jia, and J. Berant, Eds. Association for Computational Linguistics, 2018, pp. 37–46.

[10] M. Gardner, Y. Artzi, V. Basmova, J. Berant, B. Bogin, S. Chen, P. Dasigi, D. Dua, Y. Elazar, A. Gottumukkala, N. Gupta, H. Hajishirzi, G. Ilharco, D. Khashabi, K. Lin, J. Liu, N. F. Liu, P. Mulcaire, Q. Ning, S. Singh, N. A. Smith, S. Subramanian, R. Tsarfaty, E. Wallace, A. Zhang, and B. Zhou, "Evaluating models' local decision boundaries via contrast sets," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Association for Computational Linguistics, 2020, pp. 1307–1323.

[11] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of NLP models with checklist," in *Proceedings of the 2020 Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, Eds. Association for Computational Linguistics, 2020, pp. 4902–4912.

[12] C. G. Northcutt, A. Athalye, and J. Mueller, "Pervasive label errors in test sets destabilize machine learning benchmarks," *CoRR*, vol. abs/2103.14749, 2021.

[13] D. Khashabi, S. Min, T. Khot, A. Sabharwal, O. Tafjord, P. Clark, and H. Hajishirzi, "Unifiedqa: Crossing format boundaries with a single QA system," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Association for Computational Linguistics, 2020, pp. 1896–1907.

[14] Google Inc., "Google search service," http://google.com/.

[15] "Tool, replication package, and specific implementation details for this paper." [Online]. Available: https://github.com/imcsq/ASE21-QAAskeR

[16] M. Gupta, N. Kulkarni, R. Chanda, A. Rayasam, and Z. C. Lipton, "Amazonqa: A review-based question answering task," in *Proceedings of the 2019 International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 4996–5002.

[17] Q. Jin, B. Dhingra, Z. Liu, W. W. Cohen, and X. Lu, "Pubmedqa: A dataset for biomedical research question answering," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Association for Computational Linguistics, 2019, pp. 2567–2577.

[18] S. Suster and W. Daelemans, "Clicr: a dataset of clinical case reports for machine reading comprehension," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, M. A. Walker, H. Ji, and A. Stent, Eds. Association for Computational Linguistics, 2018, pp. 1551–1563.

[19] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading wikipedia to answer open-domain questions," in *Proceedings of the 2017 Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, R. Barzilay and M. Kan, Eds. Association for Computational Linguistics, 2017, pp. 1870–1879.

[20] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.

[21] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020.

[22] SuperGLUE, "Superglue benchmark leadorboard," https://super.gluebenchmark.com/leaderboard.

[23] Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu, "Attention-over-attention neural networks for reading comprehension," in *Proceedings of the 2017 Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, R. Barzilay and M. Kan, Eds. Association for Computational Linguistics, 2017, pp. 593–602.

[24] T. Schick and H. Schütze, "Exploiting cloze questions for few-shot text classification and natural language inference," *CoRR*, vol. abs/2001.07676, 2020.

[25] Z. Zhang, J. Yang, and H. Zhao, "Retrospective reader for machine reading comprehension," *CoRR*, vol. abs/2001.09694, 2020.

[26] D. Dzendzik, C. Vogel, and J. Foster, "English machine reading comprehension datasets: A survey," *CoRR*, vol. abs/2101.10421, 2021.

[27] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," in *Proceedings of the 2019 Annual Conference on Neural Information Processing Systems, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 3261–3275.

[28] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning, "Hotpotqa: A dataset for diverse, explainable multi-hop question answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018, Brussels, Belgium, October 31 - November 4, 2018*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Association for Computational Linguistics, 2018, pp. 2369–2380.

[29] D. Khashabi, S. Chaturvedi, M. Roth, S. Upadhyay, and D. Roth, "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, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, M. A. Walker, H. Ji, and A. Stent, Eds. Association for Computational Linguistics, 2018, pp. 252–262.

[30] S. Chen, S. Jin, and X. Xie, "Validation on machine reading comprehension software without annotated labels: A property-based method," in *Proceedings of the 2021 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, Athens, Greece, August 23-28, 2021*, D. Spinellis,

G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 590–602.
[31] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases." Department of Computer Science, Hong Kong University, Tech. Rep. HKUST-CS98-01, 1998.
[32] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, Jan. 2018.
[33] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011, the Ninth International Conference on Quality Software.
[34] X. Xie, Z. Zhang, T. Y. Chen, Y. Liu, P. Poon, and B. Xu, "METTLE: A metamorphic testing approach to assessing and validating unsupervised machine learning systems," *IEEE Trans. Reliab.*, vol. 69, no. 4, pp. 1293–1322, 2020.
[35] Z. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Trans. Software Eng.*, vol. 42, no. 3, pp. 264–284, 2016.
[36] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 2018 International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, M. Huchard, C. Kästner, and G. Fraser, Eds. ACM, 2018, pp. 132–142.
[37] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 2018 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. 303–314.
[38] Z. Q. Zhou and L. Sun, "Metamorphic testing of driverless cars," *Commun. ACM*, vol. 62, no. 3, pp. 61–67, 2019.
[39] S. Wang and Z. Su, "Metamorphic object insertion for testing object detection systems," in *Proceedings of the 2020 International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 2020, pp. 1053–1065.
[40] P. He, C. Meister, and Z. Su, "Structure-invariant testing for machine translation," in *Proceedings of the 2020 International Conference on Software Engineering, Seoul, ICSE 2020, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 961–973.
[41] S. Gupta, P. He, C. Meister, and Z. Su, "Machine translation testing via pathological invariance," in *Proceedings of the 2020 Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, ESEC/FSE 2020, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 863–875.
[42] P. He, C. Meister, and Z. Su, "Testing machine translation via referential transparency," in *Proceedings of the 2021 International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 410–422.
[43] B. Yan, B. Yecies, and Z. Q. Zhou, "Metamorphic relations for data validation: a case study of translated text messages," in *Proceedings of the 2019 International Workshop on Metamorphic Testing, MET@ICSE 2019, Montreal, QC, Canada, May 26, 2019*, X. Xie, P. Poon, and L. L. Pullum, Eds. IEEE / ACM, 2019, pp. 70–75.
[44] Z. Sun, J. M. Zhang, M. Harman, M. Papadakis, and L. Zhang, "Automatic testing and improvement of machine translation," in *Proceedings of the 2020 International Conference on Software Engineering, ICSE 2020, Seoul, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 974–985.
[45] Explosion, "spacy toolkit," https://spacy.io/.
[46] Universal Dependencies Contributors, "Universal dependencies," https://universaldependencies.org/.
[47] T. D. Smedt and W. Daelemans, "Pattern for python," *J. Mach. Learn. Res.*, vol. 13, pp. 2063–2067, 2012.
[48] N. Kitaev and D. Klein, "Constituency parsing with a self-attentive encoder," in *Proceedings of the 2018 Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, I. Gurevych and Y. Miyao, Eds. Association for Computational Linguistics, 2018, pp. 2676–2686.
[49] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H. Hon, "Unified language model pre-training for natural language understanding and generation," in *Proceedings of the 2019 Annual Conference on Neural Information Processing Systems, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 13 042–13 054.
[50] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Association for Computational Linguistics, 2004, pp. 74–81.
[51] R. Rehurek, "Gensim toolkit," https://radimrehurek.com/gensim/.
[52] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.
[53] S. Longpre, Y. Lu, and J. Daiber, "MKQA: A linguistically diverse benchmark for multilingual open domain question answering," *CoRR*, vol. abs/2007.15207, 2020.
[54] T. Onishi, H. Wang, M. Bansal, K. Gimpel, and D. A. McAllester, "Who did what: A large-scale person-centered cloze dataset," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, J. Su, X. Carreras, and K. Duh, Eds. The Association for Computational Linguistics, 2016, pp. 2230–2235.
[55] S. Suster and W. Daelemans, "Clicr: a dataset of clinical case reports for machine reading comprehension," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, M. A. Walker, H. Ji, and A. Stent, Eds. Association for Computational Linguistics, 2018, pp. 1551–1563.
[56] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. H. Hovy, "RACE: large-scale reading comprehension dataset from examinations," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, M. Palmer, R. Hwa, and S. Riedel, Eds. Association for Computational Linguistics, 2017, pp. 785–794.
[57] S. Eger and Y. Benz, "From hero to zéroe: A benchmark of low-level adversarial attacks," in *Proceedings of the 2020 Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing, AACL/IJCNLP 2020, Suzhou, China, December 4-7, 2020*, K. Wong, K. Knight, and H. Wu, Eds. Association for Computational Linguistics, 2020, pp. 786–803.
[58] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, M. Palmer, R. Hwa, and S. Riedel, Eds. Association for Computational Linguistics, 2017, pp. 2021–2031.