QAAskeR+: A Novel Testing Method for Question Answering Software via Asking Recursive Questions

Xiaoyuan Xie (xxie@whu.edu.cn)
Wuhan University
Shuo Jin
Wuhan University
Songqiang Chen
Wuhan University

Research Article

Keywords: question answering, testing and validation, recursive metamorphic testing, natural language processing

Posted Date: April 20th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1563040/v1

QAASKER$^+$: A Novel Testing Method for Question Answering Software via Asking Recursive Questions

Xiaoyuan Xie[1*], Shuo Jin[1] and Songqiang Chen[1*]

[1*]School of Computer Science, Wuhan University, China.

*Corresponding author(s). E-mail(s): xxie@whu.edu.cn;
i9schen@gmail.com;
Contributing authors: imjinshuo@whu.edu.cn;

Abstract

Question Answering (QA) is an attractive and challenging area in NLP community. With the development of QA technique, plenty of QA software has been applied in daily human life to provide convenient access of information retrieval. To investigate the performance of QA software, many benchmark datasets have been constructed to provide various test cases. However, current QA software is mainly tested in a reference-based paradigm, in which the expected outputs (labels) of test cases are mandatory 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 work, we propose a novel testing method, QAASKER$^+$, with five new Metamorphic Relations for QA software. QAASKER$^+$ does not refer to the annotated labels of test cases. Instead, based on the idea that a correct answer should imply a piece of reliable knowledge that always conforms with any other correct answer, QAASKER$^+$ tests QA software by inspecting its behaviors on multiple recursively asked questions that are relevant to the same or some further enriched knowledge. Experimental results show that QAASKER$^+$ can reveal quite a few violations that indicate actual answering issues on various mainstream QA software without using any pre-annotated labels.

Keywords: question answering, testing and validation, recursive metamorphic testing, natural language processing

1 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 attractive but very challenging mission that requires machine to understand the human language and infer information from it as the human do Nguyen et al (2016); Zhang et al (2020). As illustrated in Tables 2 and 3, respectively, given a question, QA software intelligently comprehends the relevant information from a huge knowledge base or one lengthy textual passage and returns the deduced answer. QA software has been widely used in daily human life to provide convenient access of information retrieval now. For instance, many intelligent devices are equipped with a virtual assistant, such as Siri from Apple and DuerOS from Baidu, which can provide the QA service.

Recently, we have seen many algorithms being proposed to improve the performance of QA software. At the same time, various benchmark datasets with different topics and task formats have been constructed to evaluate how well machine can answer textual questions by referring to the information stored in knowledge bases Berant et al (2013); Yih et al (2016); Trivedi et al (2017); Bao et al (2016) or implied in textual materials Rajpurkar et al (2018); Clark et al (2019); Kwiatkowski et al (2019); Rajpurkar et al (2016). **Nevertheless, the testing methods for QA software are still primitive and thin.** Specifically, current QA testing practices mainly adopt **the reference-based paradigm**. 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 Clark et al (2019); He et al (2018). 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. 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 replied 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 Gardner et al (2020); Ribeiro et al (2020). 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 Northcutt et al (2021).

All in all, it is of great necessity to provide a new testing method for 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 unlabeled 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 novel testing method named QAASKER$^+$ as well as five new Metamorphic Relations (MRs) to achieve this goal. QAASKER$^+$ tests QA software via recursively asking multiple questions that are relevant to the same or some further enriched knowledge. Its core idea is that a correct answer should imply a piece of reliable knowledge that always conforms with any other correct answer. More specifically, given a test input, QAASKER$^+$ first synthesizes a piece of knowledge according to the question of this (source) input and the corresponding answer (source output) from QA software. This piece of knowledge can then be optionally enriched by introducing some additional facts relevant to the answer. Afterwards, QAASKER$^+$ raises a new question (follow-up input) based on the knowledge and recursively asks QA software for the new answer (follow-up output). Finally, it checks the new answer against the knowledge. If both the source and follow-up outputs are correct, the synthesized knowledge 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 should be erroneous.

We have performed comprehensive experiments to evaluate the effectiveness of QAASKER$^+$. Specifically, we use QAASKER$^+$ to test four representative QA software that covers two mainstream types of QA software and achieves the state-of-the-art performance. The results show that QAASKER$^+$ can effectively reveal quite a few violations on each of these four test objects without referring to any annotated labels. Meanwhile, most of the revealed violations are valid to expose actual answering issues of the four tested QA software, from which multiple types of answering issues can be identified. For example, we found the QA software based on knowledge bases can both miss necessary entities and include unnecessary entities. And problems like missing answers, nonrecognition of question types, and potential generalization issues are exposed on the

QA software based on textual materials. Besides, we found that both retraining and fine-tuning QA models on the samples generated with the proposed MRs can help to repair the revealed defects. And we also preliminarily explore the usefulness of QAASKER$^+$ on a popular real-life QA application, the Google Search service.

This paper extends our preliminary work Chen et al (2021a) that is presented as a research paper at the ASE 2021 conference. Particularly, this paper enriches the existing preliminary work in the following aspects:

- We extend our methodology and formulate a new method called QAASKER$^+$. Besides recursively raising new questions strictly based on the source input and output, QAASKER$^+$ further excavates some additional facts relevant to the source output to enrich the synthesized knowledge. This helps us obtain more diverse and complicated new questions as follow-up inputs, which may contribute to a better fault detection ability. Particularly, we design and implement two new Metamorphic Relations (MRs) in QAASKER$^+$ based on this idea.
- We briefly introduce the taxonomy of QA software, where two mainstream categories of QA software are reviewed. According to the systematic taxonomy, we add three new test objects to further comprehensively evaluate the effectiveness of our method on diverse QA software. Two of them belong to the new category that has not been considered in our preliminary work.
- We perform comprehensive experiments to evaluate the effectiveness of our new method on the above test objects. We found that the new MRs have a higher violation detection rate than the existing MRs. And we also reveal many new issues and summarize the shortcomings for the added test objects. Promising results on all test objects confirm the usefulness of QAASKER$^+$ on mainstream QA software.
- We introduce a new research question to compare the issue detection ability of our recursive MRs, which take the novel idea of recursively asking, to two representative non-recursive MRs that generate semantically equivalent questions as follow-up inputs via synonym replacement and back translation. Results demonstrate the superior fault detection effectiveness of our MRs.
- We try one new repair strategy, which takes the value of recursively asking based on the knowledge inferred from the actual test output, to fix the test objects against the revealed answering issues. Results demonstrate that this method can also show a fairly good fixing effect.

In summary, the contributions of this work, which are the super-set of the contributions in our preliminary work, are as follows:

- We propose a method named QAASKER$^+$ to test **Q**uestion **A**nswering software via **R**ecursively **Ask**ing multiple questions relevant to the same or some further **enriched** knowledge. 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 five novel Metamorphic Relations in QAASKER$^+$. They generate the new question (follow-up input) on the basis of the knowledge synthesized from the existing question (source input) and the answer QA software returns for this question (source output). Two of them further enrich the knowledge with some correct information about the source output. The generated questions are of distinct types (e.g., general questions and wh-questions) or asking for different objects in the knowledge. The implementation of these MRs has been released.
- We carry out comprehensive experiments to evaluate the effectiveness of QAASKER$^+$. Results demonstrate that QAASKER$^+$ can successfully reveal many valid violations on various QA software. Moreover, we found our MRs have better fault detection effectiveness than the MRs that generate follow-up questions via synonym replacing and back translation. And we have also identified a few actual answering issues according to the revealed violations and designed methods to fix the revealed issues based on the proposed MRs, which may inspire the developers to further improve the tested QA software.
- 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 https://github.com/imjinshuo/QAAskeR-plus.

The rest of this paper is structured as follows. Section 2 states the motivation of this work. Section 3 introduces QA software and Metamorphic Testing, which are the test object and the basis of our method, respectively. Section 4 elaborates the details of QAASKER$^+$, with five novel MRs proposed. Afterwards, Section 5 and Section 6 describe the settings and the results of the evaluation on QAASKER$^+$, respectively. Next, Section 7 discusses the real-life usage of QAASKER$^+$. Section 8 presents the threats to validity and Section 9 lists the related works. Finally, Section 10 draws a conclusion and lists our future work.

2 Motivation

As introduced above, QA software has been widely used in daily human life, thus there is an urgent demand to assure the quality of its returned answers and reveal its undisclosed defects. But currently, almost all the NLP models, including the core models in QA software, are mainly tested in the reference-based paradigm Ribeiro et al (2020); Chen et al (2021b). As explained in Section 1, using this test paradigm, the testers must obtain a well-annotated benchmark dataset at first, which means that the manually annotated reference answers are mandatory during testing QA software. Once presented with the output answers to unannotated questions, current testing methods cannot automatically decide 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 story based on the QA task shown in Table 1.

Suppose that Tom is a junior entertainment editor. One day, Tom is asked to collect the information about all the recipients of the Academy Award for Best Actress over years. For convenience, Tom applies one QA software to retrieve the relevant information. For instance, to retrieve the recipient of the Academy Award for Best Actress in 1963, he inputs a question "*Which actress won the Academy Award for Best Actress in 1963?*" (Question 1). After few seconds, he receives the answer "*Anna Magnani*" (Answer 1). He collects all the results that he needs in this way. Without having the off-the-shelf ground truth labels to verify the obtained answers, Tom chooses to trust them and directly report them to his leader. Finally, he is strictly blamed because there are quite a few mistakes in his report, including wrongly taking *Anna Magnani* as the recipient of the 1963 Academy Award for Best Actress. Such an experience brings negative outcomes to Tom and greatly dampens his belief in the QA software. But if we consider Tom's question a test case, it is indeed not easy to automatically and quickly verify the output answer now, since no ground truth label is available. Therefore, **to avoid such awkward experiences 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.

Besides, 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 1, 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 Gardner et al (2020); Ribeiro et al (2020). 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 Clark et al (2019); He et al (2018). In addition, the manual annotation could also introduce some errors Northcutt et al (2021), thus hurts the accuracy of the test results.

Therefore, in this work, we aim to propose a testing method to support these requirements. The proposed method should not depend 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.

3 Background

3.1 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 Nguyen et al (2016), E-commerce services Gupta et al (2019), and healthcare Jin et al (2019); Suster and Daelemans (2018a). According to the form of the

Table 1 A Motivation Example

Question 1	Which actress won the Academy Award for Best Actress in 1963?
Answer 1	Anna Magnani
Question 2	In which years did Anna Magnani win the Academy Award for Best Actress?
Answer 2	1955
Question 3	In which years did the actress from Italy win the Academy Award for Best Actress?
Answer 3	1955 and 1961

information source for deducing answers, QA software can be firstly divided into two categories, namely the ones using information in the knowledge base (KBQA) and the other ones depending on information in the raw textual materials (TBQA) Han et al (2020). And judging from whether each question is equipped with a specified knowledge base or textual material to deduce the answer, the two categories of QA software can be further respectively divided into the closed-world ones and the open-world ones[1] Gupta et al (2019). In this work, we apply our method to test several representative QA software in these categories. In the following, we will introduce the details of these categories of QA software, especially the ones we use as the test objects in this work.

3.1.1 QA Software Based on Information in Knowledge Base

With the development of the knowledge base techniques like knowledge graph, the information about various objects can be stored in the knowledge base in a structural form. In such structural information, one object is represented as an entity and its properties are indicated by this entity's relations to other entities. In this way, large-scale information can be properly stored. At the same time, such structural information can be easily processed by machine. As a result, people are inspired to design the QA software based on the knowledge base (KBQA software) to automatically answer the textual questions raised by users according to the abundant information in knowledge base Wang et al (2020). Table 2 shows an example of a KBQA task.

Due to the large capacity of knowledge base, communities have constructed several general knowledge bases, such as Freebase Bollacker et al (2008), DBPedia Lehmann et al (2015), and Wikidata Tanon et al (2016), to hold a great number of universal knowledge on all aspects. Existing KBQA studies mainly focus on providing QA services based on these general knowledge bases. Since the information in general knowledge bases is not specified to answer any specific question, KBQA is considered to be mainly performed in the open-world manner, which means that the KBQA software requires the question as the only input and would automatically retrieve relevant information in one unified general knowledge base to answer all questions Chen et al (2017); Han et al (2020). Considering that open-world KBQA is the mainstream KBQA manner

[1]Some studies refer them as "open-domain QA" and "closed-domain QA".

Table 2 An Example of a KBQA Task

Some Key Info in Knowledge Base	"Patricia Neal".*awards_won* = {m.03mlw6_} m.03mlw6_.*award* = {"Academy Award for Best Actress"} m.03mlw6_.*year* = {"1963"}
Question	Which actress won the Academy Award for Best Actress in 1963?
Correct Answer	Patricia Neal

The information "entity1".*propertyName* = {"entity2", "entity3"} suggests the value of a property named *propertyName* of entity "entity1" contains two entities, i.e., "entity2" and "entity3".
The entities denoted as "x.xxxxxx" are the virtual compound entities synthesized by the knowledge base to represent some abstract objects.

and there are many relevant methods and datasets available, we mainly focus on the open-world KBQA software in this work.

Two main categories of methods to implement open-world KBQA software are commonly known as the semantic parsing-based ones and the information retrieval-based ones Lan et al (2021). The semantic parsing-based methods first parse a question into a query of logic form and then execute it over the knowledge database for finding the answers. When generating the search query, such methods require effectively locating the accurate search space. To tackle this problem, one state-of-the-art method in this way proposed by Lan and Jiang (2020) uses a staged approach to generate the necessary query graphs effectively. Another mainstream KBQA method is based on information retrieval. Given a question, they first retrieve a question-specific sub-graph from the knowledge database based on the identified query intent of the given questions and then apply some ranking algorithms to select the most appropriate entities as the answer. A major challenge for the retrieval-based methods is the lack of supervision signals at intermediate steps. To address this challenge, He et al (2021a) propose a method called NSM+h that adopts the teacher-student learning framework to reach the state-of-the-art. NSM+h contains a student network focusing on the KBQA task itself and a teacher network learning to provide supervision signals for improving the reasoning ability of the student network. In this work, we use these two state-of-the-art KBQA methods to build corresponding KBQA test objects as representatives.

3.1.2 QA Software Based on Information in Textual Materials

Though the structural information in knowledge base can be easily understood and processed by the machine, it has inherent limitations, such as incompleteness and fixed schemes. It is also costly to construct, maintain, and update the knowledge base. As a result, recent studies focus on another QA manner, namely TBQA, which directly extracts information from some non-structural textual materials, such as the textual passages in Wikipedia Chen et al (2017). Such QA software can effectively leverage the information in any textual materials that humans read, thus is much more flexible and extensible. Table 3 gives an example of a TBQA task. But compared to the structural information in knowledge base, the information in textual materials is fairly harder to

comprehend and extract. The comprehension on textual materials is a challenging key problem in TBQA and has attracted a lot of works to improve its performance Zhang et al (2020).

Since extracting information from textual materials is non-trivial, most of the existing TBQA studies focus on the relatively simpler closed-world manner. In this manner, a specific textual reference passage is attached with the given input question for deducing this question. TBQA software should answer this question based on the attached passage. Considering many relevant algorithms and datasets are available, we mainly target the closed-world TBQA software in this paper. In fact, the closed-world TBQA is still playing the core role in the open-world TBQA systems, which automatically retrieve relevant textual materials from web Chen et al (2017). Therefore, our method can also work on the open-world TBQA software. We briefly discuss such usages in Section 7.

Closed-world TBQA tasks can be solved with various methods. For example, we can simply further fine-tune some pre-trained language models, such as ROBERTa Liu et al (2019) and T5 Raffel et al (2020), to obtain relatively good performance on several closed-world TBQA task formats like the span extraction and the boolean judgment. Earlier studies mainly solve the TBQA tasks of different formats using distinct models; while recent works move to building unified and effective methods for the general format-agnostic TBQA systems. Khashabi et al (2020) pioneer to propose a method, UnifiedQA, which builds a single model to solve the closed-world TBQA tasks in distinct formats. UnifiedQA first trains a text-to-text model on some seed QA datasets of multiple task formats, during which the textual materials and the questions of different task formats are taken as input without using format-specific prefixes. Users should then fine-tune this pre-trained model into specialized models for better performance on the specific QA tasks. The UnifiedQA model has shown pretty promising performance on par with or better than the format-agnostic models on many benchmark datasets. Afterwards, Tafjord and Clark (2021) propose an improved version of UnifiedQA, named Macaw, to realize the versatile and zero-shot closed-world TBQA. Zero-shot means that the Macaw model can deliver fairly good performance on various datasets without being further fine-tuned on any target corpus. Macaw also first trains the model on several seed QA corpus. But it applies more training tasks, such as teaching the model to raise questions for specified answers. Besides, Macaw further fine-tunes the model on several science questions to improve its zero-shot ability. Some officially trained Macaw models are publicly released as high-quality off-the-shelf TBQA software to the community. In this work, we adopt these two state-of-the-art methods to prepare representative TBQA test objects.

3.2 Metamorphic Testing

To get rid of the dependency on the annotated labels in testing QA software, we design our testing method based on the idea of Metamorphic Testing. Metamorphic Testing (MT) is one proper candidate solution to bypass the labels of test cases, since it was proposed to reuse the passed test cases and alleviate the

Table 3 An Example of a TBQA Task

Textual Material	... the 13th season of "SuperNatural", an American dark fantasy television series created by Eric Kripke, premiered on October 12, 2017, on the CW ...
Question	Whose new episode comes out on October 12?
Correct Answer	SuperNatural

oracle problem during software testing Chen et al (1998, 2018). 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 Xie et al (2011) and the unsupervised clusters Xie et al (2020). And Zhou et al (2016) adopt 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 Zhang et al (2018); Tian et al (2018); Zhou and Sun (2019); Wang and Su (2020) and language translation services He et al (2020); Gupta et al (2020); He et al (2021b); Yan et al (2019); Sun et al (2020, 2022).

4 Methodology

4.1 A Recursive Metamorphic Testing Method for QA Software

Let us revisit the motivating example in Table 1. Suppose that Jack and Merry are another two senior entertainment editors. They also ask the same question (Question 1) as Tom does and get the same answer (Answer 1) from QA software. They do not have the ground truth label to verify this output answer as well. But unlike Tom, by seeing this answer, Jack further asks the QA software one new question "*In which years did Anna Magnani win the Academy Award for Best Actress?*" (Question 2). For this question, the QA software replies "*1955*" (Answer 2). At the same time, by further considering a relevant fact that Anna Magnani is from Italy, Merry also asks a new question, "*In which years did the actress from Italy win the Academy Award for Best Actress?*" (Question 3). And she obtains the answer "*1955 and 1961*" (Answer 3). By comparing Answer 1 and Answer 2, Jack is then confused. And Merry also feels

something wrong when comparing Answer 1 and Answer 3. This is because "*1963*" is not included in their second answer as they have expected. But the good thing is: even if Jack and Merry may not be clear about which answer is wrong, they have found some clues saying that the QA software is not reliable and at least one of the returned answers that they receive is incorrect.

The operation of Jack and Merry in the above example just illustrates the basic idea of the method we propose in this paper. To break the dependency on the annotated labels in testing QA software, we propose a method named QAASKER$^+$. Its core idea is to test QA software via Recursively Asking multiple relevant questions and check the relation among the output answers for these questions. The input of QAASKER$^+$ is the QA software under test (SUT) and a list of unlabeled questions, but no manually annotated ground truth labels 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 the ground truth label. And the MRs in QAASKER$^+$ is based on the idea that a correct answer should imply a piece of reliable knowledge that always conforms with any other correct answer.

More specifically, given an input question q (known as the "source input"), let us denote the answer that SUT replies for q as a (known as the "source output"). Then, a piece of knowledge k can be synthesized based on q and a. Taking Question 1 and Answer 1 in Table 1 as the example, by synthesizing Question 1 and Answer 1, QAASKER$^+$ can obtain a piece of knowledge "*Anna Magnani won the Academy Award for Best Actress in 1963*". Obviously, if the answer a is correct, then the knowledge k is a true fact that should always hold. Thereby, **on the basis of this synthesized knowledge** k, QAASKER$^+$ next **recursively raises one new question** q' **(known as one follow-up input) relevant to** k. Let us denote the answer that SUT replies for q' as a' (known as the corresponding follow-up output). As mentioned above, if a' is also correct, it should conform with the above knowledge k, regarding q'. Otherwise, a violation is revealed to indicate an answering error, because at least one of a and a' should be wrong.

Just like what Jack and Merry have respectively done in the above example, QAASKER$^+$ raises the new question in two ways. Following Jack's operation, QAASKER$^+$ can directly raise a new question based on k. In the above example, Question 2 "*In which years did Anna Magnani win the Academy Award for Best Actress?*" can just be considered as directly constructed based on k. And obviously, Answer 2 "*1955*" does not conform with k considering Question 2. One answering error is thereby revealed. Besides, QAASKER$^+$ can also follow Merry's operation to further enrich k with a piece of correct information (also considered as one fact) about the source output before raising the new question. In the above example, the information "*Anna Magnani is from Italy*" is adopted to enrich k. And the enriched knowledge k^+ "*An actress from*

Italy won the Academy Award for Best Actress in 1963" is thereby obtained. Then, Question 3 "*In which years did the actress from Italy win the Academy Award for Best Actress?*" can just be considered as constructed based on k^+. Similarly, Answer 3 "*1955 and 1961*" does not conform with k^+ as well as k regarding Question 3, thus an answering error is revealed.

We can see that the above process does not require the label of q. Instead, our recursive MT method QAASKER$^+$ would automatically formulate oracles to inspect the test outputs. As a result, such a method **effectively breaks the dependency on the manually annotated labels during testing** and therefore provides a possible solution to testing QA software on unlabeled data.

Based on the above idea of recursively asking, in QAASKER$^+$, we design five novel MRs by considering the consistency among the input question and output answer pairs related to the same or some further enriched 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. Among the proposed MRs, three MRs directly generate a new question based on the synthesized knowledge and the other two MRs raise a new question based on a piece of further enriched knowledge. QAASKER$^+$ realizes these MRs with three components, 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 components in detail.

4.2 Proposed Metamorphic Relations

As mentioned above, in this paper, we propose five MRs based on the idea of recursively asking (we call them "recursive MRs"). They test the QA software by checking its behaviors on multiple recursively asked questions that are relevant to the same or some further enriched knowledge. In this section, we introduce the overall idea of each MR. The specific implementation of these MRs would be introduced in detail in the following sections. To simplify the demonstration, we denote the source and follow-up input question 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 4.

We first introduce the three MRs that generate the new questions directly based on the synthesized knowledge.

MR1: Answering a new follow-up wh-question that is raised based on the knowledge regarding one existing source wh-question and its source output answer.

This MR is eligible for the test inputs with a wh-question on which SUT's output is not "<*NoAnswer*>". As shown in Figure 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} and perform the output checking. 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.

MR2: Answering a new follow-up general question that is raised based on the knowledge regarding one existing source wh-question and its source output answer.

This MR is also eligible for the test cases whose question is one wh-question with a non-"<*NoAnswer*>" SUT output. Figure 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 that the a^t_{GEN} should be "Yes". We then run SUT with q'_{GEN} to obtain a'_{GEN} and perform the output checking. 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}.

MR3: Answering a new follow-up wh-question that is generated based on the knowledge regarding one existing general or alternative question and its source output answer.

This MR is eligible for the test inputs that have a general question or an alternative question. As shown in Figure 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}. Next, we perform the output checking. 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.

The other two MRs that generate a new question according to the further enriched knowledge are based on MR1 and MR2, respectively. They enrich the synthesized knowledge in MR1 and MR2. In this paper, we solely consider enriching the synthesized knowledge in MR1 and MR2, since it is fairly easier to extract abundant information about the output answer of the wh-questions. We leave the task of enriching the knowledge regarding more types of questions as our future work.

Table 4 Definition of Question Types

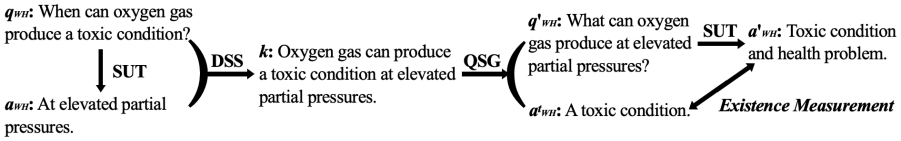
Abbr.	Type	Examples
WH	wh-question	Q: *Who was Emma's brother?* A: *Duke Richard II.* Q: *How many soldiers were in each Tumen?* A: *10,000.*
GEN	general question	Q: *Is this the last year for once upon a time?* A: *Yes.* Q: *Does a cow have to be pregnant to lactate?* A: *No.*
ALT	alternative question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

MR1+: Answering a new follow-up wh-question that is generated based on the knowledge regarding one existing source wh-question and some extra information about its source output answer.

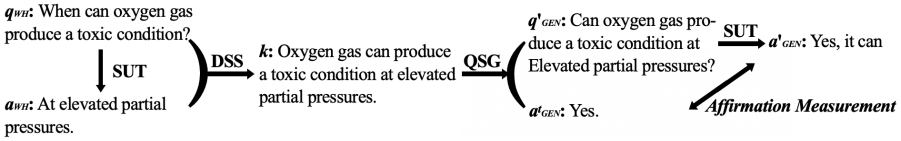
Just like MR1, MR1+ is also eligible for the test inputs with a wh-question on which SUT's output is not <*NoAnswer*>. And as shown in Figure 1(d), the overall process of MR1+ is similar to that of MR1. The difference is that we leverage DSS to synthesize a **further enriched knowledge** in MR1+. More specifically, for a wh-question q_{WH} and the SUT's output a_{WH}, we first leverage DSS to retrieve a piece of correct information about a_{WH} from the given knowledge base or textual materials. Next, we use DSS to synthesize an enriched knowledge k^+ based on q_{WH} and the retrieved correct information about a_{WH}. Afterwards, we adopt QSG to generate a new wh-question q'_{WH} and its target answer a^t_{WH} based on this enriched knowledge k^+. The remained steps are the same with those in MR1. We run SUT with q'_{WH} to obtain a'_{WH} and perform the output checking. If the SUT is correct, both a_{WH} and a'_{WH} should be correct, and the enriched knowledge k^+ is a true fact. Since a^t_{WH} is deduced from k^+, a^t_{WH} should be 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.

MR2+: Answering a new follow-up general question that is raised based on the knowledge regarding one existing source wh-question and some extra information about its source output answer.

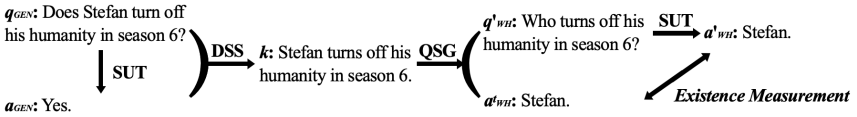
MR2+ is also eligible for the test cases whose question is a wh-question where SUT gives a non-<*NoAnswer*> output. As shown in Figure 1(e), similar to MR1+ and MR1, MR2+ differs from MR2 in the synthesis of knowledge. More specifically, for a wh-question q_{WH} and the SUT's output a_{WH}, we adopt the similar operation in MR1+ to collect a piece of correct information about a_{WH} and synthesize an enriched knowledge k^+ based on q_{WH} and the collected correct information about a_{WH}. Next, we apply QSG to generate a new general question q'_{GEN} against k^+. The remained steps are the same with those in MR2. The expected target answer a^t_{GEN} for q'_{GEN} is "Yes". We run SUT with q'_{GEN} to obtain a'_{GEN} and perform the output checking. 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}.



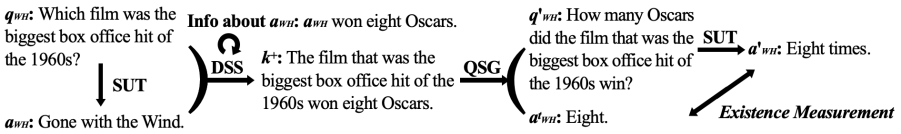
(a) Test Process of MR1



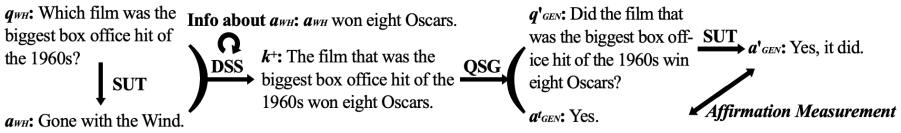
(b) Test Process of MR2



(c) Test Process of MR3



(d) Test Process of MR1+



(e) Test Process of MR2+

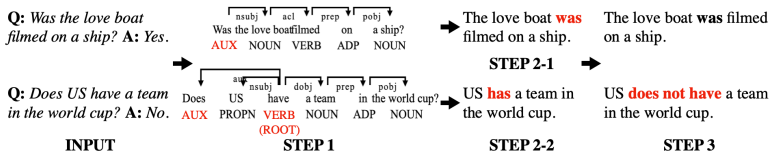
Fig. 1 Proposed Recursive Metamorphic Relations

4.3 Declarative Sentence Synthesis

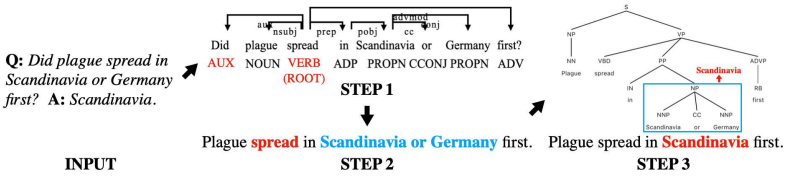
In this section, we introduce the methods to synthesize the declarative sentence (the knowledge k or the enriched knowledge k^+) from a pair of question and SUT's corresponding output answer. The question could be one of three types, namely general questions, alternative questions, and wh-questions.

4.3.1 Declarative Sentence Synthesis Based on General Question and Its Answer

For a general question q_{GEN} and SUT's answer a_{GEN}, three steps are needed to synthesize the corresponding declarative sentence k. Figure 2(a) shows this



(a) Declarative Sentence Synthesis Based on General Question and Its Answer

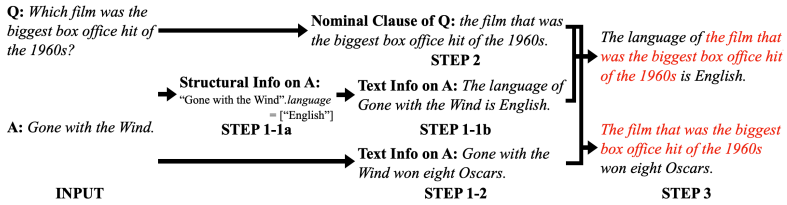


(b) Declarative Sentence Synthesis Based on Alternative Question and Its Answer

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) Heuristic Rules for Declarative Sentence Synthesis Based on Wh-Question and Its Answer



(d) Declarative Sentence Synthesis Based on Wh-Question and Extra Information about Its Answer

Fig. 2 Process and Example of Declarative Sentence Synthesis

process. Specifically, we first use spaCy toolkit[2] to analyze the token dependency and locate the auxiliary (AUX)[3] 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

[2] https://spacy.io/

[3] 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 https://universaldependencies.org/.

with Pattern Library Smedt and Daelemans (2012) (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}.

4.3.2 Declarative Sentence Synthesis Based on Alternative Question and Its Answer

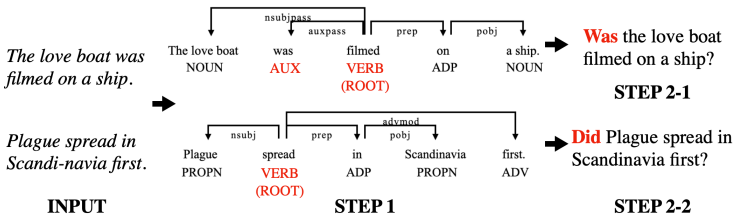
It also involves three steps to synthesize the declarative sentence from the given general question q_{ALT} and SUT's answer a_{ALT}. The whole process is shown in Figure 2(b). The first two steps are similar to the operations in Section 4.3.1. The difference is that the obtained k after step 2 still contains the alternatives (text in blue). So we adopt the Berkeley Neural Parser Kitaev and Klein (2018) 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}.

4.3.3 Declarative Sentence Synthesis Based on Wh-Question and Its Answer

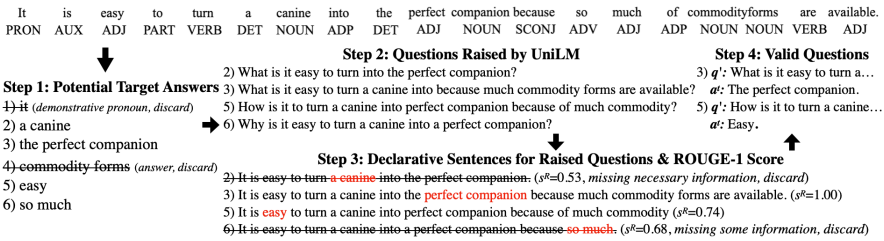
To obtain a fairly reliable declarative sentence from the given wh-question q_{WH} and SUT's answer a_{WH}, we design numerous heuristic rules to process every distinct form of q_{WH}. Due to the limited space in this paper, we only list four basic operations in Figure 2(c). The detailed rules (e.g.,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

4.3.4 Declarative Sentence Synthesis Based on Wh-Question and Extra Information about Its Answer

Given a wh-question q_{WH} and SUT's answer a_{WH}, three steps are required to synthesize the declarative sentence of the enriched knowledge with respect to q_{WH} and a piece of correct information about a_{WH}. Figure 2(d) presents the overall process. The first step is to retrieve a piece of correct information about a_{WH}. Consider that the knowledge bases or textual materials should store a great number of facts about a_{WH}. For KBQA software, we seek its knowledge base for such information. We first locate a_{WH} and randomly pick one of its properties as such information (step 1-1a). Next, we use some heuristic rules to generate a sentence to describe the picked information (step 1-1b). And for TBQA software, we directly pick a sentence including a_{WH} from the textual material as such information (step 1-2). After obtaining the correct information about a_{WH}, the second step for testing both KBQA and TBQA software is to transform q_{WH} into a nominal clause (step 2). We also prepare quite a few heuristic rules to achieve this transformation. Finally, we substitute a_{WH} in the sentence obtained in the first step with the nominal clause obtained in the second step (step 3). After that, the declarative sentence of the enriched



(a) General Question Sentence Generation



(b) Wh-Question Sentence Generation

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

knowledge is obtained. The detailed heuristic rules in the above steps can be found in our online supplementary material.

4.4 Follow-up Question Sentence Generation

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

4.4.1 General Question Sentence Generation

The generation of general questions could be seen as the opposite process to Section 4.3.1. As shown in Figure 3(a), given a declarative sentence k, we first locate the predictive verb (VERB(ROOT)) in k (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 Smedt and Daelemans (2012) 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).

4.4.2 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. Figure 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. 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 in our online supplementary material).

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 Dong et al (2019), instead of designing complicated heuristic rules. UniLM has shown fairly promising performance of question generation on SQuAD1 dataset Rajpurkar et al (2016). 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 Lin (2004), to provide a similarity score[4] 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).

4.5 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 4.2, 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

[4]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).

Table 5 Example of Existence Measurement

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

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".

4.5.1 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 5 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 Řehůřek and Sojka (2010). After that, the maximum similarity for each word in a^t_{WH} is calculated as shown in the right-most column. With this 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 predefined 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.

4.5.2 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 6 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 6, 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 $s^{aff} > \theta^{aff}$.

5 Experimental Setup

5.1 Research Questions

To evaluate QAASKER$^+$, we study five research questions:

RQ1: *The overall effectiveness of* QAASKER$^+$. In this RQ, we aim to give a global picture on the effectiveness of QAASKER$^+$ in revealing the issues of various QA software without using the annotated ground truth labels. And we also discuss the violation detection ability of our MRs and the performance of four test objects based on the test results.

RQ2: *Effectiveness comparison with non-recursive Metamorphic Relations.* In this paper, we design five novel recursive MRs that generate the follow-up input questions based on both the source input question and the source output answer. We are interested in whether these recursive MRs have better fault detection ability than those non-recursive MRs. So, in this RQ, we compare the effectiveness of our five recursive MRs with two representative non-recursive MRs that generate semantically equivalent questions as follow-up inputs.

RQ3: *Validity of the revealed violations.* Considering the imperfection in most of the NLP generation and measurement methods Dong et al (2019); Lin (2004), it is meaningful to understand the factuality of the revealed violations. Therefore, in this RQ, we perform a deeper inspection on the revealed violations to measure their validity.

RQ4: *Analysis on the revealed true violations.* To provide an intuitive and constructive impression on the issues revealed by our method, 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.

RQ5: *Helpfulness to Fix the Answering Issues Revealed by MRs.* In many studies that propose MT-based testing methods for deep learning software, researchers generate new samples with the proposed MRs to retrain or fine-tune the models, so as to fix the defects revealed by MRs. We are also interested in whether our MRs are helpful to fix the revealed issues. Therefore, in this RQ, we study the performance of the models that are "fixed" by the proposed MRs in two manners.

5.2 Data Preparation

To evaluate the effectiveness of QAASKER$^+$ on KBQA and TBQA software, we first collect proper datasets as the source of test cases. For KBQA software, we choose two most widely-used benchmark datasets, namely WebQuestionSP and ComplexWebQ.

- **WebQuestionSP** is a classic KBQA benchmark dataset with wh-questions collected via the Google Suggest API. All its questions can be answered with entities retrieved from the Freebase knowledge base. Most of its questions can be answered by performing simple reasoning on Freebase. It includes 3k question-answer pairs for training and 1.6k for testing.
- **ComplexWebQ** is a more challenging version of WebQuestionsSP, whose questions are further complicated. Similarly, all these complicated questions are wh-questions and can be answered based on Freebase. But they require more complex reasoning on Freebase to find out the answer. ComplexWebQ has 31.1k samples for training and 3.5k samples for testing.

For TBQA software, we leverage three typical benchmark datasets, namely SQuAD2, BoolQ, and NatQA. These datasets cover the mainstream types of TBQA questions and tasks.

- **SQuAD2** is a span extraction dataset, where the answer of each question is a span of words from the textual material 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 Khashabi et al (2020)). 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 textual material. We use the version provided by UnifiedQA where each question is appended with a referential textual material. 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 run the corresponding KBQA or TBQA software under test to obtain the source outputs of all the test cases in above datasets. After that, we apply each of the five MRs on its eligible source test samples, respectively. The testing (i.e., violation measurement) is then conducted on the eligible samples.

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

5.3 Test Objects

As introduced in Section 3.1, in this work, we build our KBQA and TBQA test objects using four representative state-of-the-art methods, namely NSM+h He et al (2021a), Multi-hop Complex KBQA Lan and Jiang (2020), UnifiedQA Khashabi et al (2020), and Macaw Tafjord and Clark (2021). The first two are for KBQA test objects while the latter two are for TBQA test objects. In this section, we introduce how we prepare the test objects.

Among the four state-of-the-art QA methods, NSM+h, Multi-hop Complex KBQA, and Macaw have already publicly released their models for the above benchmark datasets. Therefore, we directly load the corresponding official models as our test objects[6]. Meanwhile, UnifiedQA only releases its pre-trained base model and requires us to fine-tune this base model on our target dataset. Considering that UnifiedQA can solve multiple QA tasks with a unified model, we apply it to train one general model based on all three datasets as our test object. Specifically, we collect the training samples from SQuAD2, BoolQ, and NatQA to form a hybrid training set with 236,422 samples. And following Khashabi et al (2020), we fine-tune the pre-trained T5-large-based UnifiedQA model on this hybrid training set and regard the optimal checkpoint as the final test object. During training process, the batch size is 3, the learning rate is 2e-5, the loss accumulates every 5 steps, the validation is conducted per 5000 steps, and the early stop tolerance is 10 times.

6 Results and Analysis

6.1 RQ1: The Overall Effectiveness of QAAskeR$^+$

To evaluate the overall effectiveness of QAAskeR$^+$ in revealing the answering issues, we report the violation rates (the ratio of the violated test cases in all eligible test cases) of four test objects at each MR and dataset. Tables 7 and 8 respectively show the violation rates of two KBQA software and the other two TBQA software. Based on our test results, we discuss the fault detection ability of different MRs, as well as summarize the characteristics of the performance for each test object.

Table 7 Violation Rates of Two KBQA Test Objects against Our MRs

Object	Dataset	MR1	MR1+
NSM+h	WebQuestionSP	73.04%	91.96%
	ComplexWebQ	71.06%	86.36%
Multi-hop Complex KBQA	WebQuestionSP	61.43%	84.11%
	ComplexWebQ	86.11%	88.21%

[6]The official models can be found at their replication packages as follows:
NSM+h: https://github.com/RichardHGL/WSDM2021_NSM.
Multi-hop Complex KBQA: https://github.com/lanyunshi/Multi-hopComplexKBQA.
Macaw: https://github.com/allenai/macaw.

Considering that two KBQA test objects are only able to answer the wh-questions, we solely test them with MR1 and MR1+ that generate new wh-questions based on existing wh-questions. From Table 7, we found that both MR1 and MR1+ have revealed quite a few violations on every dataset. This demonstrates the effectiveness of QAASKER$^+$ to expose the answering issues on KBQA software. When comparing the violations revealed by different MRs, we can see that MR1+, which is newly designed in this work to further extend and complicate the question, has revealed more violations than MR1. This indicates that generating more diverse and complicated questions as follow-up inputs tends to have a better fault detection ability. And we also compare the performance of two KBQA test objects and found that there is no essential difference between their performance. NSM+h triggers more violations on the relatively easier WebQuestionSP; while shows slightly better performance on the fairly more complicated ComplexWebQ.

Table 8 Violation Rates of Two TBQA Test Objects against Our MRs

Object	Dataset	MR1	MR2	MR3	MR1+	MR2+
UnifiedQA	SQuAD2	37.05%	65.85%	90.91%	61.11%	92.35%
	BoolQ	–	–	72.78%	–	–
	NatQA	51.98%	96.92%	46.15%	62.40%	99.69%
Macaw	SQuAD2	65.61%	56.16%	93.94%	74.34%	81.88%
	BoolQ	–	–	70.18%	–	–
	NatQA	59.73%	25.96%	73.33%	78.70%	38.81%

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

The two general TBQA test objects can answer wh-questions, general questions, and alternative questions. Therefore, we apply all five MRs to test them. From Table 8, we can also see that all MRs have revealed many violations on all datasets. This demonstrates the effectiveness of QAASKER$^+$ for the TBQA software. And the new MR1+ and MR2+, which introduce more diverse and complex follow-up inputs, are also found to reveal more violations than the original MR1-MR3. This again confirms the effectiveness and superiority of our idea to enrich and complicate the input question for detecting more errors. Furthermore, we found that MR2 and MR3, which involve the transformation of question types, have revealed relatively more violations than MR1 on both test objects. This is interesting because UnifiedQA and Macaw have shown promising ability to solve wh-questions and general questions on SQuAD2 and BoolQ, respectively, according to the results of reference-based tests Khashabi et al (2020); Tafjord and Clark (2021). But according to our results, they fail to return proper answers to the general question and wh-question related to very similar knowledge on these two datasets. From this point, we conjecture that UnifiedQA and Macaw might overfit the training samples, thus 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 and Macaw to figure out the questions of distinct types across datasets, which is vital in unifying the solutions for different TBQA task formats Khashabi et al (2020).

We also compare the performance of UnifiedQA and Macaw according to our test results. We see that UnifiedQA triggers fewer violations than Macaw on MR1, MR3, and MR1+, which require test objects to answer wh-questions. Meanwhile, Macaw shows better performance on MR2 and MR2+ that require test objects to answer general questions. Considering that the general questions are fairly rare in datasets, such results indicate that UnifiedQA, which requires being fine-tuned on the target dataset, can better process the test cases familiar in the dataset; while is weaker at the relatively rare ones. But Macaw, which provides zero-shot QA service on any target dataset, delivers fairly consistent but also limited performance across the test cases with distinct patterns. This suggests us to use Macaw to provide a preliminary QA service when the training data of expected patterns is limited; while fine-tune UnifiedQA on sufficient training data of expected patterns for better performance.

6.2 RQ2: Effectiveness Comparison with Non-Recursive Metamorphic Relations

To understand whether the idea of recursively asking can contribute to better fault detection performance, we would like to compare the effectiveness of our recursive MRs with several non-recursive MRs. However, there is no existing non-recursive MRs for the QA software addressing various kinds of questions. Inspired by our previous work of testing the machine reading comprehension software that only solves the general questions Chen et al (2021b), we design two non-recursive MRs to generate semantically equivalent questions as follow-up inputs for the QA software addressing various kinds of questions. These two non-recursive MRs generate follow-up input questions via synonym replacement (SR) and back translation (BT). Specifically, we have:

- $\mathbf{MR}_{SR}$: Following Chen et al (2021b), given a source input question q, we replace all the adjectives in q with their corresponding synonyms to obtain the follow-up question q'. As a consequence, the semantics of q and q' are the same. Therefore, for q', this MR expects the SUT to give an answer that is consistent with the answer of q. The synonyms are obtained according to the widely-used WordNet dictionary.
- $\mathbf{MR}_{BT}$: Besides synonym replacement, Liu et al (2021) apply back translation to obtain the semantically equivalent texts to test the dialogue systems. We borrow such idea to devise this MR. Given a source input question q, we first translate q into its Chinese expression and then translate it back into the English expression to obtain the follow-up question q'. The semantics of q and q' are the same as well, thus this MR also expects a consistent answer. The translation is automatically performed with Baidu translation service.

As a reminder, following Chen et al (2021b) and Liu et al (2021), we have not introduced other special designs into MR_{SR} and MR_{BT} to automatically check the quality of the new questions generated by them.

Table 9 Violation Rates of Four Test Objects against Two Non-Recursive MRs

Object	Dataset	MR_{SR}	MR_{BT}
NSM+h	WebQuestionSP	52.15%	43.05%
	ComplexWebQ	38.59%	43.97%
Multi-hop Complex KBQA	WebQuestionSP	20.43%	16.21%
	ComplexWebQ	13.86%	29.00%
UnifiedQA	SQuAD2	25.70%	20.08%
	BoolQ	18.46%	10.49%
	NatQA	32.64%	22.18%
Macaw	SQuAD2	37.59%	44.03%
	BoolQ	15.75%	27.68%
	NatQA	46.56%	53.25%

We apply MR_{SR} and MR_{BT} to test the four test objects on all datasets as well. Table 9 presents the test results when using these two non-recursive MRs. Compared to the test results of using our five recursive MRs in Tables 7 and 8, we found that MR_{SR} and MR_{BT} give much lower violation rates than our five recursive MRs in most cases. This demonstrates that our MRs have fairly superior fault detection effectiveness.

We conjecture that the superior fault detection effectiveness of our recursive MRs is due to the higher execution dissimilarity between the source and follow-up test cases. Chen et al (2003) have revealed that the MRs leading to dissimilar execution manners tend to have a higher fault detection ability. When using QAASKER$^+$, given one question q, any of our five recursive MRs generates a new question q'_R with different types or to ask for different objects. This makes the execution of QA software to deduce the answer for q'_R tend to be different from that for q. As a consequence, if the execution of q is faulty, the execution of q'_R tends to bypass the same fault to get a different answer and triggers the violation. And when the execution of q is correct, the execution of q'_R then tries more execution manners to touch the fault. This is also more likely to trigger violations. By contrast, the two non-recursive MRs generate a new question q'_N, which has a fairly similar query pattern with q. As a result, if the execution of q is faulty, the execution of q'_N may also be trapped in a similar fault. An output with a similar error may thereby be obtained and the violation would not be triggered. Besides, when the execution of q is correct, the execution of q'_N is not to touch the other faulty execution manner as well.

Table 10 Validity Rate of the Inspected Violations Revealed on NSM+h

Dataset	MR1	MR1+	$\mathbf{MR}_{SR}$	$\mathbf{MR}_{BT}$
WebQuestionSP	82%	81%	84%	75%
ComplexWebQ	80%	85%	83%	61%

6.3 RQ3: Validity of the Revealed Violations

By having revealed quite a few violations on all the four test objects in RQ1 and outperformed the non-recursive MRs with large margins in RQ2, we are particularly interested in evaluating the validity of the revealed violations. We perform a manual inspection on the revealed violations. Specifically, if there is 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 Dong et al (2019); Lin (2004). 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.

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. Since the implementation of our method and its actual operation for all KBQA software and all TBQA software are the same, respectively, we pick NSM+h and UnifiedQA as representatives and investigate the factuality of the revealed violations on them to bypass repetitive workload that should lead to similar conclusions. For either NSM+h or UnifiedQA, given an MR and a dataset, if more than 100 violations are revealed, the inspectors examine the validity of the randomly picked 100 violations. Otherwise, they will check the validity of all the violations.

After the inspection on the randomly picked violations, we perform Cohen's Kappa statistics Cohen (1960). The agreement rate between two inspectors is quite perfect (0.87) 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 and RQ2 is convincing, and most of the revealed violations are meaningful and should be seriously considered by the developers and the users of QA software.

We first analyze the inspection result for our MRs. The inspection results over the randomly picked violations revealed on NSM+h and UnifiedQA are presented in the 2nd to 3rd columns of Table 10 and the 2nd to 6th columns of Table 11, respectively. For the representative of KBQA test objects, NSM+h,

Table 11 Validity Rate of the Inspected Violations Revealed on UnifiedQA

Dataset	MR1	MR2	MR3	MR1+	MR2+	MR_{SR}	MR_{BT}
SQuAD2	81%	100%	90%	73%	73%	84%	66%
BoolQ	–	–	87%	–	–	88%	63%
NatQA	85%	100%	83%	75%	76%	85%	62%

we found that over 80% of the inspected violation revealed by both MR1 and MR1+ are valid. Meanwhile, for the representative of TBQA test objects, UnifiedQA, we found that the inspected violations revealed by MR2 are all valid, over 80% of the inspected violations revealed by MR1 and MR3 are valid, and more than 70% of the inspected violations revealed by MR1+ and MR2+ are valid. These valid rates are considered to be acceptable when compared with the precision in similar testing methods for the machine translation software Gupta et al (2020); He et al (2020, 2021b). The valid rates for MR1+ and MR2+ on UnifiedQA are slightly lower. We consider this is mainly because it is more challenging to extract proper (clear and accurate) additional information about source output from the textual materials. It is not easy to appropriately supplement the information with various expressions in textual materials into the preliminary knowledge as well. And we also review the invalid violations for the other MRs and found that they mainly result from the limitation of the semantic equivalence measurement and the question generation on a 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 demonstrate the reliability of the design and implementation in our QAASKER$^+$ regarding the quality of wh-question generation and semantics similarity measurement.

And we also compare the valid rate of the inspected violations revealed by our MRs to that of the two non-recursive MRs. The corresponding inspection results on the inspected violations revealed by MR_{SR} and MR_{BT} on NSM+h and UnifiedQA are presented in the last two columns of Tables 10 and 11, respectively. We first notice that the valid rate of the inspected violations revealed by MR_{BT} is much lower than that of our MRs. We found this is mainly because the translation service sometimes gives imperfect questions that are different from the original questions with respect to their semantics. Since we cannot expect the correct relation between the outputs for the original and these imperfect new questions, a few invalid violations would be revealed on these imperfect inputs. Meanwhile, we design specific methods to examine the quality of the generated questions when implementing our MRs. Therefore, our MRs generate fewer imperfect new questions and lead to fairly fewer invalid violations. As a reminder, according to the result of RQ2, MR_{BT} is weaker

Table 12 Number of Erroneous Answers on True Violations of NSM+h

Dataset	MR1	MR1+
WebQuestionSP	48% , 52%	48% , 52%
ComplexWebQ	86% , 14%	88% , 12%

(Please refer to the footnotes in Table 13)

Table 13 Number of Erroneous Answers on True Violations of UnifiedQA

Dataset	MR1	MR2	MR3	MR1+	MR2+
SQuAD2	27% , 73%	25% , 75%	44% , 56%	31% , 69%	32% , 68%
BoolQ	–	–	21% , 79%	–	–
NatQA	52% , 48%	58% , 42%	0% , 100%	49% , 51%	66% , 34%

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.

than our MRs regarding the violation detection effectiveness as well. And as for MR_{SR}, we found that the valid rate of the inspected violations revealed by it is slightly better than our MRs. We consider this is mainly because the synonym replacement is relatively easier and does not change the original question a lot, thus is less likely to generate many imperfect questions. But as we discussed in RQ2, this largely limits the violation detection effectiveness of MR_{SR} as well.

6.4 RQ4: Analysis on the Revealed True Violations

In this RQ, we further study the valid true violations revealed by our method, which are identified in RQ3. Specifically, we first locate the erroneous answers (that is, the source or the follow-up answer) and next summarize the answering issues of two representative test objects.

We first identify the erroneous answer in the true violations. Tables 12 and 13 present the statistics results for NSM+h and UnifiedQA, respectively. We found that QAASKER$^+$ can find errors on both source test cases and follow-up test cases for two test objects. More specifically, for NSM+h, we notice that more violations are blamed for the source output answer on ComplexWebQ dataset than on WebQuestionSP dataset. This is reasonable because the questions in ComplexWebQ are more challenging than those in WebQuestionSP. And for UnifiedQA, 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 generated by MR2 and MR3 are of the fairly unfrequent formats in SQuAD2 and BoolQ, respectively, this further supports our conjecture about the limited generalization of UnifiedQA on question types across datasets, which has been discussed in Section 6.1.

And to give a systematic and intuitive understanding about the issues that QAASKER$^+$ reveal on the test objects, we further identify and summarize the answering issues according to the revealed true violations. We first analyze

the violations revealed on NSM+h. Since the output of the KBQA software is a set of several entities, there could only be three types of possible errors in returned answer, namely giving totally wrong entities, missing necessary entities, and returning extra entities. All these three types of errors have been identified in the revealed violations on NSM+h. The corresponding examples are listed in Table 14. As a reminder, these three types of errors may happen on one case simultaneously. Here we only present the examples on which only one type of error happens for the clear demonstration.

Giving totally wrong entities. We first found that NSM+h sometimes can return some totally wrong answers, which means that there is no intersection between the expected and returned answers. This indicates that NSM+h almost misunderstands the intent of the corresponding input questions at all. For instance, in Example N-1, the question asks for the team that Adrian Peterson plays for in his college years. However, NSM+h wrongly returns the team that he currently serves as a professional athlete.

Missing necessary entities. We also notice that NSM+h may miss some necessary entities and give an incomplete answer, where the output answer is the proper subset of the expected answer. For instance, in Example N-2, the information in the knowledge base clearly indicates that Margaret Hoover has been educated in two institutions. However, NSM+h only retrieves one of them as the final answer.

Returning extra entities. Besides, even having correctly returned all the necessary entities, NSM+h may further give some extra unnecessary entities. In Example N-3, the question only asks for the city of Acadia University. However, NSM+h seems not to accurately parse the border of this query. It returns not only the correct answer of city "Wolfville", but also the unnecessary entity "Nova Scotia", which is the province where Acadia University locates.

Compared to the KBQA tasks whose answer is made up of several entities, the answers for the TBQA tasks are more flexible since their answers are in the form of text without much restriction. As a result, UnifiedQA suffers from more types of issues. We identify and summarize five types of answering issues on UnifiedQA as follows. The corresponding examples are listed in Table 15.

***<NoAnswer>* for answerable questions.** We first found that UnifiedQA cannot answer some answerable questions. In Example U-1, the textual material 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 U-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 U-2-2 should be either "Yes" or "No", but SUT wrongly gives an irrelevant verb as the answer.

Table 14 Examples of Revealed Answering Issues on NSM+h

#	Example N-1	Example N-2	Example N-3
Knowledge Base Info	"Adrian Peterson".*sport_league_draft* = {"m.02h7kn3"} "m.02h7kn3".*school* = {"University of Oklahoma"} "Adrian Peterson".*pro_athlete_team* = {"m.03gkk7c"} "m.03gkk7c".*team* = {"Minnesota Vikings"}	"Margaret Hoover".*education* = {"m.0n1gxgt", "g.11c3kps_b6"} "m.0n1gxgt".*institution* = {"Davidson College"} "g.11c3kps_b6".*institution* = {"Bryn Mawr College"}	"Acadia University".*headquarters* = {"m.05sq0xg"} "m.05sq0xg".*citytown* = {"Wolfville"} "m.05sq0xg".*state_province_region* = {"Nova Scotia"}
Question	What team did Adrian Peterson play for in college?	Where did Margaret Hoover go to college?	What city is Acadia University in?
Expected	{"University of Oklahoma"}	{"Bryn Mawr College", "Davidson College"}	{"Wolfville"}
NSM+h	{"Minnesota Vikings"}	{"Davidson College"}	{"Wolfville", "Nova Scotia"}

Table 15 Examples of Revealed Answering Issues on UnifiedQA

#	Example U-1	Example U-2-1	Example U-2-2
Textual Material	... The IPCC receives funding ... 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, 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	IPCC's deputy secretary	The Carrie Diaries	Yes
UnifiedQA	*<NoAnswer>*	No	Instigated

#	Example U-3	Example U-4	Example U-5
Textual Material	... Some broadcasts are free ... some are encrypted and require a monthly subscription ...	... 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	Some encrypted broadcasts	The parent company of NDS	Shi Tianze's father
UnifiedQA	Sky	The name of the company that	His father

Irrelevant content of the answer. Although successful in recognizing the type of the question, the UnifiedQA model sometimes gives answers with irrelevant content. Example U-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 U-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 U-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 textual material.

6.5 RQ5: Helpfulness to Fix the Answering Issues Revealed by MRs

In this RQ, we investigate the helpfulness of our method in fixing the issues revealed by MRs for the test objects. In existing studies on MT for deep learning software, there are two manners to fix the revealed issues with the proposed MRs, namely to retrain new models Tian et al (2018); Liu et al (2021) and to fine-tune the existing models He et al (2020); Wang and Su (2020) using the samples generated with MRs, respectively. In our preliminary work, we have investigated the effectiveness of retraining new models with three preliminary MRs, i.e., MR1-MR3. In this work, we first report the effectiveness of retraining manner with all five MRs. Next, we further investigate the effectiveness of fine-tuning manner. In this RQ, we solely conduct the evaluation on NSM+h and UnifiedQA as representatives as well, because only they release friendly replication packages to perform retraining and fine-tuning.

Retraining new models from scratch on the **training cases** expanded by the proposed MRs is a widely-used manner to fix the revealed issues on deep learning models in many studies Tian et al (2018); Liu et al (2021). Following the existing works that use this method, we expand the training samples with the five proposed MRs to retrain new NSM+h and UnifiedQA models. The performance difference between the original and new models would demonstrate the helpfulness of the MRs to fix the answering issues in a retraining manner.

Specifically, for each training sample in a dataset, we adopt the MRs that are eligible on it to generate corresponding new training samples. Considering that the ground truth labels for training samples should have existed for performing supervised training, we decide to directly use the ground truth label of every training sample to synthesize the knowledge for generating new training samples. For every dataset, we collect the generated samples together with all samples in the original training set to form a new training set. The new training set of WebQuestionSP and ComplexWebQ for NSM+h include 5,967 and 33,740 samples, respectively, and the new hybrid training set for UnifiedQA contains 545,157 samples in total. After that, we retrain several new models with the corresponding new training sets, during which the hyper parameters keep the same as introduced in Section 5.3. Finally, we test each retrained model on the corresponding test sets that are also kept the same as before.

The test results for the new retrained NSM+h and UnifiedQA models are presented in Tables 16 and 17, respectively. We first found that the reduction in violation rates is substantial against all MRs for both test objects. For UnifiedQA, the improvement on MR2 and MR3, which involve the transformation of question types, is especially significant. These findings indicate that the proposed MRs can help to eliminate many answering issues revealed on the test objects by retraining them on the expanded training samples.

Fine-tuning the original models with the new samples generated from the existing **test cases** is a new manner to repair the revealed issues in deep learning models in recent studies He et al (2020); Wang and Su (2020). It is less resource- and time-intensive than retraining from scratch He et al (2021b).

Table 16 Performance of the Retrained NSM+h Model

Dataset	MR1	MR1+
WebQuestionSP	28.30% *(44.74%)*	46.40% *(45.56%)*
ComplexWebQ	32.66% *(38.40%)*	67.31% *(19.05%)*

Values in brackets indicate the improvement to the corresponding performance in Table 7.

Table 17 Performance of the Retrained UnifiedQA Model

Dataset	MR1	MR2	MR3	MR1+	MR2+
SQuAD2	24.86% *(12.19%)*	0.15% *(65.70%)*	44.44% *(46.47%)*	31.51% *(29.60%)*	0.48% *(91.87%)*
BoolQ	–	–	26.74% *(46.04%)*	–	–
NatQA	21.26% *(30.72%)*	0.00% *(96.92%)*	33.33% *(12.82%)*	47.25% *(15.15%)*	0.00% *(99.69%)*

Values in brackets indicate the improvement to the corresponding performance in Table 8.

In this work, we also investigate if our MRs can help to fix the revealed issues using the new samples generated from the test cases. In particular, considering the labels of test cases are usually unavailable, we try directly generating new samples based on the knowledge synthesized from the actual source outputs, which is with exactly the same process of performing testing using QAASKER$^+$. The performance difference between the original and fine-tuned models would show the helpfulness of our MRs to fix the revealed issues via fine-tuning.

Specifically, given a dataset, we first divide its test cases into two subsets at the ratio of 1:1. We use the test cases in one subset (denoted as the fine-tuning subset) to generate the new samples for fine-tuning; while leave the test cases in the other subset (denoted as the testing subset) for testing. Next, for each sample in the fine-tuning subset, we leverage the MRs that are eligible on it to generate the new samples. Since there are usually no easily accessible ground truth labels for test cases, we directly generate new samples based on the knowledge synthesized with the actual outputs of models. This helps us efficiently leverage the unlabeled test cases to fix the answering issues. And we also collect the new samples together with all the original samples in the fine-tuning subset to form the final fine-tuning set. The final fine-tuning set of WebQuestionSP and ComplexWebQ for NSM+h include 1,735 and 3,059 samples, respectively, and the hybrid fine-tuning set for UnifiedQA contains 28,866 samples in total. Afterwards, we fine-tune each model with the corresponding fine-tuning set for 20 epoches and collect the optimal model during the fine-tuning process as the final test object. Finally, we test each fine-tuned model on the corresponding testing subset.

The test results for the fine-tuned NSM+h model and UnifiedQA model are presented in Tables 18 and 19, respectively. We surprisingly found that the violation rates of both test objects have reduced a lot as well. More specifically, the improvement at MR1, MR2, MR3, and MR2+ is generally comparable to that under the setup of retraining. The improvement at MR1+ is relatively less but still considerable. These suggest that fine-tuning the existing models with the samples generated based on the knowledge regarding the actual outputs

Table 18 Performance of the Fine-tuned NSM+h Model

Dataset	MR1	MR1+
WebQuestionSP	24.61% *(47.93%)*	63.71% *(28.19%)*
ComplexWebQ	23.32% *(47.25%)*	51.46% *(36.72%)*

Values in brackets indicate the improvement to the performance of original model on the test subsets.

Table 19 Performance of the Fine-tuned UnifiedQA Model

Dataset	MR1	MR2	MR3	MR1+	MR2+
SQuAD2	23.79% *(8.59%)*	0.28% *(70.08%)*	45.45% *(40.26%)*	57.14% *(17.86%)*	1.03% *(95.30%)*
BoolQ	–	–	27.74% *(47.47%)*	–	–
NatQA	25.22% *(28.28%)*	0.00% *(96.72%)*	44.44% *(12.70%)*	50.28% *(14.10%)*	0.00% *(99.84%)*

Values in brackets indicate the improvement to the performance of original model on the test subsets.

can also obtain a fairly good repairing effect against the revealed answering issues, moreover, in a less resource- and time-intensive manner.

To conclude, both retraining new models and fine-tuning existing models with the samples generated with our MRs can help to fix the revealed issues and reduce the violations. However, we could find that there still exist many violations on the retrained or fine-tuned models. 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 can also argue that our method is necessary for the correctness inspection of QA software output and the in-depth problem revealing of QA software. This again confirms the significance of QAASKER$^+$ as a testing method.

7 Discussion on Real-life Usage

With the development of QA algorithms some industrial products have been able to provide preliminary QA services. For example, the Google Search service[7] 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 referential textual material 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 TBQA software**.

According to our observation, the Google Search service can mainly answer wh-questions now. Therefore, we only try MR1 and 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, for each MR, we first randomly choose 20 wh-questions from an open-world TBQA benchmark,

[7] https://www.google.com/

MKQA[8] Longpre et al (2020). These 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 test cases.

Finally, 5 out of 20 test cases regarding MR1[9] and 7 out of 20 test cases regarding MR1+[10] trigger the violation. Let us present one of the violations revealed by each MR in detail. For MR1, 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. For MR1+, we first query "*What was the first movie to have color?*" and get the source answer "*A Visit to the Seaside*" from Google Search. With the introduction materials about this movie provided by Wikipedia, we learn that the director of "*A Visit to the Seaside*" is "*George Albert Smith*". Then, we recursively query "*Who is* ***the director of*** *the first movie to have color?*" on Google search but obtain an unexpected answer "*Cecil B. DeMille*", which triggers a violation. Actually, the answer to the source question should be "*La Vie et la passion de Jésus Christ*". These demonstrate that QAASKER$^+$ finds true erroneous answers returned by the Google Search service. In a word, this trial demonstrates the potential of QAASKER$^+$ to reveal real-life bugs for daily QA applications. Moreover, we consider it has also suggested that people can leverage our methodology of recursively asking to perform a "just-in-time test" during their usage of QA software, so as to obtain some clues about the correctness of the returned answer.

8 Threats to Validity

The first threat to validity is about the representativeness of the test objects and the datasets. In this work, we apply our method to test four test objects and one real-life application. These four test objects have covered both mainstream categories of QA software and achieve the state-of-the-art performance. And Google Search is a popular and typical QA application practically used in human life. Thus, we consider they are suitable representative test objects and their test results can reflect the effectiveness of our QAASKER$^+$ in general. Actually, QAASKER$^+$ can be considered as a black-box testing method that only involves the input and output of QA software and an arbitrary fact about the output. Therefore, QAASKER$^+$ should be generalizable to any other

[8] All the questions in MKQA can be answered with the public knowledge from the web.
[9] Based on the search results obtained on April 10, 2021.
[10] Based on the search results obtained on April 8, 2022.

QA software of these mainstream QA manners, just using the way in which we test the representative objects. 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 He et al (2021a); Khashabi et al (2020). Since we evaluate QAASKER$^+$ on all of them, we consider the evaluation should have fairly good generalization.

The second threat to validity is about the tools that we adopt to realize the proposed MRs. As illustrated in Section 6.3, 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 70% of the inspected violations are valid. This is acceptable when compared to other MT-based test methods for DL software Gupta et al (2020); He et al (2020, 2021b). 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 quite perfect (0.87). And all the disagreements are settled after their discussion.

9 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.

9.1 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 Zhang et al (2020); Yani and Krisnadhi (2021); Dzendzik et al (2021). For KBQA, there are various benchmark datasets with different requirements. Some datasets solely contain questions which can be answered by deducing a simple relation in knowledge base Chandar et al (2016); Azmy et al (2018) and some datasets include questions that request complex reasoning to answer Bao et al (2016); Trivedi et al (2017). For TBQA, benchmark datasets are with diverse forms of task, including to fill in the blanks Onishi et al (2016); Suster and Daelemans (2018b), judge the correct options Clark et al (2019); Khashabi et al (2018); Lai et al (2017), extract the relevant spans Rajpurkar et al (2016, 2018); Yang et al (2018), and return fluent text answers Kwiatkowski et al (2019); He et al (2018). There are also datasets of the samples with adversarial inputs, such as typos Eger and Benz (2020) and irrelevant sentences Jia and Liang (2017), to test the robustness of QA software. But as mentioned in

Section 1, 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.

9.2 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 (2018) and Zhang et al (2018) propose to test AD against the relation among the steering angles under distinct weather conditions. Zhou and Sun (2019) combine MT and fuzzing and take the LiDAR point-cloud data of AD into consideration during testing. Wang and Su (2020) 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 He et al (2020), pathological invariance Gupta et al (2020), referential transparency He et al (2021b), etc. In addition to being adopted to test NMT services, MT is also found to be helpful in assessing the quality of input data Yan et al (2019) and repairing the erroneous translations Sun et al (2020, 2022) for NMT services.

In this paper, we also leverage MT to test a hot DL application, QA software. Specifically, we propose five novel MRs against the consistency among the input question and output answer pairs that are related to the same or some further enriched 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 Chen et al (2021b). 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 investigated TBQA boolean question task, but also the TBQA span extraction and free-form answering

tasks. Furthermore, we investigate the effectiveness of QAASKER$^+$ on another mainstream category of QA software, the KBQA software, and further explore the real-life usefulness of QAASKER$^+$ on the Google Search service and its helpfulness to repair the revealed issues.

10 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 five novel recursive Metamorphic Relations. QAASKER$^+$ tests QA software by checking its behaviors on multiple recursively asked questions that are relevant to the same or some further enriched knowledge. It cuts off 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$^+$ by using it to test four representative state-of-the-art QA software that covers two mainstream types of QA software, as well as a popular real-life QA application, the Google Search service. Comprehensive results demonstrate that QAASKER$^+$ can reveal quantities of valid violations that depict diverse answering issues for various kinds of mainstream QA software. Besides, we also found that our recursive MRs have a better fault detection effectiveness than two representative non-recursive MRs and can even help to fix the revealed 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.

Acknowledgments. We first sincerely appreciate the positive acknowledgment and the very kind suggestions from the anonymous reviewers for both our conference paper and this extended journal paper. 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

Azmy M, Shi P, Lin J, et al (2018) Farewell freebase: Migrating the simplequestions dataset to dbpedia. In: Bender EM, Derczynski L, Isabelle P (eds) Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018. Association for Computational Linguistics, pp 2093–2103, URL https://aclanthology.org/C18-1178/

Bao J, Duan N, Yan Z, et al (2016) Constraint-based question answering with knowledge graph. In: Calzolari N, Matsumoto Y, Prasad R (eds) COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan. ACL, pp 2503–2514, URL https://aclanthology.org/C16-1236/

Berant J, Chou A, Frostig R, et al (2013) Semantic parsing on freebase from question-answer pairs. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL. ACL, pp 1533–1544, URL https://aclanthology.org/D13-1160/

Bollacker KD, Evans C, Paritosh PK, et al (2008) Freebase: a collaboratively created graph database for structuring human knowledge. In: Wang JT (ed) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008. ACM, pp 1247–1250, https://doi.org/10.1145/1376616.1376746

Chandar S, Ahn S, Larochelle H, et al (2016) Hierarchical memory networks. CoRR abs/1605.07427. URL http://arxiv.org/abs/1605.07427, https://arxiv.org/abs/1605.07427

Chen D, Fisch A, Weston J, et al (2017) Reading wikipedia to answer open-domain questions. In: Barzilay R, Kan M (eds) Proceedings of the 2017 Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers. Association for Computational Linguistics, pp 1870–1879, https://doi.org/10.18653/v1/P17-1171

Chen S, Jin S, Xie X (2021a) Testing your question answering software via asking recursively. In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021. IEEE, pp 104–116, https://doi.org/10.1109/ASE51524.2021.9678670

Chen S, Jin S, Xie X (2021b) Validation on machine reading comprehension software without annotated labels: A property-based method. In: Spinellis

D, Gousios G, Chechik M, et al (eds) 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. ACM, pp 590–602, https://doi.org/10.1145/3468264.3468569

Chen TY, Cheung SC, Yiu SM (1998) Metamorphic testing: a new approach for generating next test cases. Tech. Rep. HKUST-CS98-01, Department of Computer Science, Hong Kong University

Chen TY, Kuo F, Tse TH, et al (2003) Metamorphic testing and beyond. In: Proceedings of the 2003 International Workshop on Software Technology and Engineering Practice, 19-21 September 2003, Amsterdam, The Netherlands. IEEE Computer Society, pp 94–100, https://doi.org/10.1109/STEP.2003.18, URL https://doi.org/10.1109/STEP.2003.18

Chen TY, Kuo FC, Liu H, et al (2018) Metamorphic testing: A review of challenges and opportunities. ACM Computing Surveys 51(1). https://doi.org/10.1145/3143561

Clark C, Lee K, Chang M, et al (2019) Boolq: Exploring the surprising difficulty of natural yes/no questions. In: Burstein J, Doran C, Solorio T (eds) 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). Association for Computational Linguistics, pp 2924–2936, https://doi.org/10.18653/v1/n19-1300

Cohen J (1960) A coefficient of agreement for nominal scales. Educational and psychological measurement 20(1):37–46

Dong L, Yang N, Wang W, et al (2019) Unified language model pre-training for natural language understanding and generation. In: Wallach HM, Larochelle H, Beygelzimer A, et al (eds) Proceedings of the 2019 Annual Conference on Neural Information Processing Systems, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp 13,042–13,054

Dzendzik D, Vogel C, Foster J (2021) English machine reading comprehension datasets: A survey. CoRR abs/2101.10421. https://arxiv.org/abs/2101.10421

Eger S, Benz Y (2020) From hero to zéroe: A benchmark of low-level adversarial attacks. In: Wong K, Knight K, Wu H (eds) 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. Association for Computational Linguistics, pp 786–803

Gardner M, Artzi Y, Basmova V, et al (2020) Evaluating models' local decision boundaries via contrast sets. In: Cohn T, He Y, Liu Y (eds) Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020. Association for Computational Linguistics, pp 1307–1323, https://doi.org/10.18653/v1/2020.findings-emnlp.117

Gupta M, Kulkarni N, Chanda R, et al (2019) Amazonqa: A review-based question answering task. In: Kraus S (ed) Proceedings of the 2019 International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. ijcai.org, pp 4996–5002, https://doi.org/10.24963/ijcai.2019/694

Gupta S, He P, Meister C, et al (2020) Machine translation testing via pathological invariance. In: Devanbu P, Cohen MB, Zimmermann T (eds) 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. ACM, pp 863–875, https://doi.org/10.1145/3368089.3409756

Han J, Cheng B, Wang X (2020) Open domain question answering based on text enhanced knowledge graph with hyperedge infusion. In: Cohn T, He Y, Liu Y (eds) Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020, Findings of ACL, vol EMNLP 2020. Association for Computational Linguistics, pp 1475–1481, https://doi.org/10.18653/v1/2020.findings-emnlp.133

He G, Lan Y, Jiang J, et al (2021a) Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In: Lewin-Eytan L, Carmel D, Yom-Tov E, et al (eds) WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021. ACM, pp 553–561, https://doi.org/10.1145/3437963.3441753

He P, Meister C, Su Z (2020) Structure-invariant testing for machine translation. In: Rothermel G, Bae D (eds) Proceedings of the 2020 International Conference on Software Engineering, Seoul, ICSE 2020, South Korea, 27 June - 19 July, 2020. ACM, pp 961–973, https://doi.org/10.1145/3377811.3380339

He P, Meister C, Su Z (2021b) 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, pp 410–422, https://doi.org/10.1109/ICSE43902.2021.00047

He W, Liu K, Liu J, et al (2018) Dureader: a chinese machine reading comprehension dataset from real-world applications. In: Choi E, Seo M, Chen

D, et al (eds) Proceedings of 2018 the Workshop on Machine Reading for Question Answering@ACL 2018, Melbourne, Australia, July 19, 2018. Association for Computational Linguistics, pp 37–46, https://doi.org/10.18653/v1/W18-2605

Jia R, Liang P (2017) Adversarial examples for evaluating reading comprehension systems. In: Palmer M, Hwa R, Riedel S (eds) Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017. Association for Computational Linguistics, pp 2021–2031, https://doi.org/10.18653/v1/d17-1215

Jin Q, Dhingra B, Liu Z, et al (2019) Pubmedqa: A dataset for biomedical research question answering. In: Inui K, Jiang J, Ng V, et al (eds) 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. Association for Computational Linguistics, pp 2567–2577, https://doi.org/10.18653/v1/D19-1259

Khashabi D, Chaturvedi S, Roth M, et al (2018) Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In: Walker MA, Ji H, Stent A (eds) 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). Association for Computational Linguistics, pp 252–262, https://doi.org/10.18653/v1/n18-1023

Khashabi D, Min S, Khot T, et al (2020) Unifiedqa: Crossing format boundaries with a single QA system. In: Cohn T, He Y, Liu Y (eds) Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020. Association for Computational Linguistics, pp 1896–1907, https://doi.org/10.18653/v1/2020.findings-emnlp.171

Kitaev N, Klein D (2018) Constituency parsing with a self-attentive encoder. In: Gurevych I, Miyao Y (eds) Proceedings of the 2018 Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers. Association for Computational Linguistics, pp 2676–2686, https://doi.org/10.18653/v1/P18-1249

Kwiatkowski T, Palomaki J, Redfield O, et al (2019) Natural questions: a benchmark for question answering research. Trans Assoc Comput Linguistics 7:452–466

Lai G, Xie Q, Liu H, et al (2017) RACE: large-scale reading comprehension dataset from examinations. In: Palmer M, Hwa R, Riedel S (eds) Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017. Association for Computational Linguistics, pp 785–794, https://doi.org/10.18653/v1/d17-1082

Lan Y, Jiang J (2020) Query graph generation for answering multi-hop complex questions from knowledge bases. In: Jurafsky D, Chai J, Schluter N, et al (eds) Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020. Association for Computational Linguistics, pp 969–974, https://doi.org/10.18653/v1/2020.acl-main.91

Lan Y, He G, Jiang J, et al (2021) A survey on complex knowledge base question answering: Methods, challenges and solutions. In: Zhou Z (ed) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021. ijcai.org, pp 4483–4491, https://doi.org/10.24963/ijcai.2021/611

Lehmann J, Isele R, Jakob M, et al (2015) Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web 6(2):167–195. https://doi.org/10.3233/SW-140134

Lin CY (2004) ROUGE: A package for automatic evaluation of summaries. In: Text Summarization Branches Out. Association for Computational Linguistics, pp 74–81

Liu Y, Ott M, Goyal N, et al (2019) Roberta: A robustly optimized BERT pretraining approach. CoRR abs/1907.11692. https://arxiv.org/abs/1907.11692

Liu Z, Feng Y, Chen Z (2021) Dialtest: automated testing for recurrent-neural-network-driven dialogue systems. In: Cadar C, Zhang X (eds) Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021. ACM, pp 115–126, https://doi.org/10.1145/3460319.3464829

Longpre S, Lu Y, Daiber J (2020) MKQA: A linguistically diverse benchmark for multilingual open domain question answering. CoRR abs/2007.15207. https://arxiv.org/abs/2007.15207

Nguyen T, Rosenberg M, Song X, et al (2016) MS MARCO: A human generated machine reading comprehension dataset. In: Besold TR, Bordes A, d'Avila Garcez AS, et al (eds) 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, CEUR Workshop Proceedings, vol 1773. CEUR-WS.org

Northcutt CG, Athalye A, Mueller J (2021) Pervasive label errors in test sets destabilize machine learning benchmarks. CoRR abs/2103.14749. https://arxiv.org/abs/2103.14749

Onishi T, Wang H, Bansal M, et al (2016) Who did what: A large-scale person-centered cloze dataset. In: Su J, Carreras X, Duh K (eds) Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016. The Association for Computational Linguistics, pp 2230–2235, https://doi.org/10.18653/v1/d16-1241

Raffel C, Shazeer N, Roberts A, et al (2020) Exploring the limits of transfer learning with a unified text-to-text transformer. J Mach Learn Res 21:140:1–140:67

Rajpurkar P, Zhang J, Lopyrev K, et al (2016) Squad: 100, 000+ questions for machine comprehension of text. In: Su J, Carreras X, Duh K (eds) Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016. The Association for Computational Linguistics, pp 2383–2392, https://doi.org/10.18653/v1/d16-1264

Rajpurkar P, Jia R, Liang P (2018) Know what you don't know: Unanswerable questions for squad. In: Gurevych I, Miyao Y (eds) Proceedings of the 2018 Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers. Association for Computational Linguistics, pp 784–789, https://doi.org/10.18653/v1/P18-2124

Řehůřek R, Sojka P (2010) Software Framework for Topic Modelling with Large Corpora. http://is.muni.cz/publication/884893/en

Ribeiro MT, Wu T, Guestrin C, et al (2020) Beyond accuracy: Behavioral testing of NLP models with checklist. In: Jurafsky D, Chai J, Schluter N, et al (eds) Proceedings of the 2020 Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020. Association for Computational Linguistics, pp 4902–4912, https://doi.org/10.18653/v1/2020.acl-main.442

Smedt TD, Daelemans W (2012) Pattern for python. J Mach Learn Res 13:2063–2067

Sun Z, Zhang JM, Harman M, et al (2020) Automatic testing and improvement of machine translation. In: Rothermel G, Bae D (eds) Proceedings of the 2020 International Conference on Software Engineering, ICSE 2020, Seoul, South Korea, 27 June - 19 July, 2020. ACM, pp 974–985, https://doi.org/10.1145/3377811.3380420

Sun Z, Zhang JM, Xiong Y, et al (2022) Improving machine translation systems via isotopic replacement. In: Damian D, Zeller A (eds) Proceedings of the 2022 International Conference on Software Engineering, ICSE 2022, Pittsburgh, USA, 21 May - 29 May, 2022. ACM, https://doi.org/10.1145/3510003.3510206

Suster S, Daelemans W (2018a) Clicr: a dataset of clinical case reports for machine reading comprehension. In: Walker MA, Ji H, Stent A (eds) 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). Association for Computational Linguistics, pp 1551–1563, https://doi.org/10.18653/v1/n18-1140

Suster S, Daelemans W (2018b) Clicr: a dataset of clinical case reports for machine reading comprehension. In: Walker MA, Ji H, Stent A (eds) 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). Association for Computational Linguistics, pp 1551–1563, https://doi.org/10.18653/v1/n18-1140

Tafjord O, Clark P (2021) General-purpose question-answering with macaw. CoRR abs/2109.02593. https://arxiv.org/abs/2109.02593

Tanon TP, Vrandecic D, Schaffert S, et al (2016) From freebase to wikidata: The great migration. In: Bourdeau J, Hendler J, Nkambou R, et al (eds) Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016. ACM, pp 1419–1428, https://doi.org/10.1145/2872427.2874809

Tian Y, Pei K, Jana S, et al (2018) Deeptest: automated testing of deep-neural-network-driven autonomous cars. In: Chaudron M, Crnkovic I, Chechik M, et al (eds) Proceedings of the 2018 International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. ACM, pp 303–314, https://doi.org/10.1145/3180155.3180220

Trivedi P, Maheshwari G, Dubey M, et al (2017) Lc-quad: A corpus for complex question answering over knowledge graphs. In: d'Amato C, Fernández M, Tamma VAM, et al (eds) The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25,

2017, Proceedings, Part II, Lecture Notes in Computer Science, vol 10588. Springer, pp 210–218, https://doi.org/10.1007/978-3-319-68204-4_22

Wang S, Su Z (2020) 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, pp 1053–1065, https://doi.org/10.1145/3324884.3416584

Wang X, Zhao S, Han J, et al (2020) Modelling long-distance node relations for KBQA with global dynamic graph. In: Scott D, Bel N, Zong C (eds) Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020. International Committee on Computational Linguistics, pp 2572–2582, https://doi.org/10.18653/v1/2020.coling-main.231, URL https://doi.org/10.18653/v1/2020.coling-main.231

Xie X, Ho JW, Murphy C, et al (2011) Testing and validating machine learning classifiers by metamorphic testing. Journal of Systems and Software 84(4):544–558. https://doi.org/https://doi.org/10.1016/j.jss.2010.11.920, the Ninth International Conference on Quality Software

Xie X, Zhang Z, Chen TY, et al (2020) METTLE: A metamorphic testing approach to assessing and validating unsupervised machine learning systems. IEEE Trans Reliab 69(4):1293–1322. https://doi.org/10.1109/TR.2020.2972266

Yan B, Yecies B, Zhou ZQ (2019) Metamorphic relations for data validation: a case study of translated text messages. In: Xie X, Poon P, Pullum LL (eds) Proceedings of the 2019 International Workshop on Metamorphic Testing, MET@ICSE 2019, Montreal, QC, Canada, May 26, 2019. IEEE / ACM, pp 70–75, https://doi.org/10.1109/MET.2019.00018

Yang Z, Qi P, Zhang S, et al (2018) Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In: Riloff E, Chiang D, Hockenmaier J, et al (eds) Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018, Brussels, Belgium, October 31 - November 4, 2018. Association for Computational Linguistics, pp 2369–2380, https://doi.org/10.18653/v1/d18-1259

Yani M, Krisnadhi AA (2021) Challenges, techniques, and trends of simple knowledge graph question answering: A survey. Inf 12(7):271. https://doi.org/10.3390/info12070271, URL https://doi.org/10.3390/info12070271

Yih W, Richardson M, Meek C, et al (2016) The value of semantic parse labeling for knowledge base question answering. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers.

The Association for Computer Linguistics, https://doi.org/10.18653/v1/p16-2033

Zhang M, Zhang Y, Zhang L, et al (2018) Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In: Huchard M, Kästner C, Fraser G (eds) Proceedings of the 2018 International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018. ACM, pp 132–142, https://doi.org/10.1145/3238147.3238187

Zhang Z, Zhao H, Wang R (2020) Machine reading comprehension: The role of contextualized language models and beyond. CoRR abs/2005.06249. https://arxiv.org/abs/2005.06249

Zhou Z, Xiang S, Chen TY (2016) Metamorphic testing for software quality assessment: A study of search engines. IEEE Trans Software Eng 42(3):264–284. https://doi.org/10.1109/TSE.2015.2478001

Zhou ZQ, Sun L (2019) Metamorphic testing of driverless cars. Commun ACM 62(3):61–67. https://doi.org/10.1145/3241979