QATest: A Uniform Fuzzing Framework for Question Answering Systems

Zixi Liu
zxliu@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yang Feng*
fengyang@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yining Yin
ynyin@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Jingyu Sun
MF21320132@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Zhenyu Chen
zychen@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Baowen Xu
bwxu@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

ABSTRACT

The tremendous advancements in deep learning techniques have empowered question answering(QA) systems with the capability of dealing with various tasks. Many commercial QA systems, such as Siri, Google Home, and Alexa, have been deployed to assist people in different daily activities. However, modern QA systems are often designed to deal with different topics and task formats, which makes both the test collection and labeling tasks difficult and thus threats their quality.

To alleviate this challenge, in this paper, we design and implement a fuzzing framework for QA systems, namely `QATest`, based on the metamorphic testing theory. It provides the first uniform solution to generate tests with oracle information automatically for various QA systems, such as machine reading comprehension, open-domain QA, and QA on knowledge bases. To further improve testing efficiency and generate more tests detecting erroneous behaviors, we design N-Gram coverage and perplexity priority based on the features of the question data to guide the generation process. To evaluate the performance of `QATest`, we experiment with it on four QA systems that are designed for different tasks. The experiment results show that the tests generated by `QATest` detect hundreds of erroneous behaviors of QA systems efficiently. Also, the results confirm that the testing criteria can improve test diversity and fuzzing efficiency.

*Yang Feng is the corresponding author.

ASE '22, October 10–14, 2022, Rochester, MI, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3556929

KEYWORDS

fuzz testing, automated testing, question answering systems, natural language processing

ACM Reference Format:
Zixi Liu, Yang Feng, Yining Yin, Jingyu Sun, Zhenyu Chen, and Baowen Xu. 2022. QATest: A Uniform Fuzzing Framework for Question Answering Systems. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3551349.3556929

1 INTRODUCTION

With the development of Natural Language Processing (NLP) [15] and deep learning techniques [22], various types of question answering (QA) systems have been designed and deployed to assist in many tasks, such as customer service, voice control, home kit, and other fields [27]. Even though QA systems have achieved great success in processing these tasks, as one kind of software program, QA systems also suffer from software bugs, which may result in plenty of losses. In recent years, many commercial question answering systems have been complained about their erroneous behaviors [1, 6], and have caused misunderstandings, or even political conflicts [19]. For example, Amazon's smart speaker Alexa could be fooled by some corner input cases to make creepy laughs, which scares the elderly and children and affects their lives [1].

However, the nature and design of modern QA systems bring many challenges to their testing tasks. First, to achieve outstanding performances under the different application scenarios, modern QA systems are often constructed with diverse architectures, such as question answering systems based on machine reading comprehension (MRC) [7], open-domain question answering (ODQA) [54], question answering on knowledge bases (KBQA) [33] and so on. These QA systems are designed to handle different sources and formats of data, which naturally makes the test data collection and labeling process difficult.

Unlike traditional software, which is often built upon logic rules programmed by developers, modern QA systems are mostly implemented based on deep learning models and algorithms that require a large number of labeled question and answer pairs to learn decision logics [32, 55]. Although some QA systems achieve high accuracy on the benchmark dataset, they may still make erroneous behaviors in the real application scenarios because of the data distribution shifts [23, 28, 40]. Meanwhile, generating tests for QA systems usually costs quite a lot of human effort to label the question data collected from usage scenarios. Moreover, another challenge of testing these QA systems lies in the construction of the test oracle, which describes the correct output for a given test input. For a given input question, the correct response of the question answering system may not be deterministic, for example, a question may correspond to multiple possible correct answers [47]. Therefore, it is difficult to define strict oracles for such systems.

To assure and improve the quality of QA systems, researchers have proposed a few automated testing techniques [13, 14]. However, they are often designed for some specific question type and thus cannot provide comprehensive support for the testing of different types of QA systems. For example, Chen et al. [14] proposed a property-based method by designing some metamorphic relations that require transforming both the question and the corresponding answer, which can only be applied to QA systems that deal with YES/NO answers. Another recent approach named QAAskeR [13] converts known seed question-answer pairs into declarative sentences, and then generates new test pairs based on the declarative sentence. However, it fails to test the QA systems designed for generating multiple reasonable answers or no answers for a given question. Moreover, all existing QA system testing approaches failed to provide testing guidance and selection strategies to improve the efficiency and effectiveness of the test generation process. This shortage results in plenty of redundant or ineffective tests, and thus lowers the testing efficiency.

Therefore, a uniform fuzzing framework that can generate tests effectively, as well as efficiently, for various types of QA systems is in dire need. This paper proposes a fuzzing framework, namely `QATest`, for mainstream types of QA systems, including KBQA, ODQA, and MRC. Specifically, `QATest` can generate a large number of tests based on a series of metamorphic transformations that are designed for all types of question answers. To improve the testing efficiency and effectiveness, we design two testing criteria to guide the test generation based on the text input domain coverage and word transition probability. Moreover, we apply a seed selection strategy to ensure the diversity of the generated question-answer pairs. We implement `QATest` and evaluate it on four QA systems, covering the widely-used ODQA, MRC, and KBQA, and we found hundreds of erroneous behaviors in these systems. Further, the experiment results show that `QATest` with seed updating and selection strategies can generate more failed tests compared with directly applying metamorphic relations. Meanwhile, the number of failed tests and diversity of the test set generated by `QATest` is higher than those of existing QA system testing methods. Moreover, we conducted a case study by selecting an open-source QA system to further analyze the bug types detected by `QATest`.

The contributions of this paper can be summarized as follows.

- **Framework.** We design the first uniform fuzzing framework, namely `QATest`, to generate massive tests with oracle information for QA systems with various architectures, including ODQA, MRC, and KBQA systems. We have implemented `QATest` and made source code publicly available[1].
- **Testing criteria.** We design two testing criteria, i.e., N-Gram coverage and perplexity priority, as guidance for the test generation process. These two criteria are specifically designed for QA systems and can improve the testing efficiency and effectiveness of various QA systems.
- **Study.** We conduct extensive studies to investigate the effectiveness and efficiency of `QATest`. We further analyze the results and summarize the common errors of QA systems.

2 BACKGROUND

The QA system is an information retrieval system that can accept users' questions described in natural language, and can find or infer users' answers from a large amount of heterogeneous data [16]. QA systems can be classified in various ways, such as based on different knowledge sources or different question types. The knowledge sources can be classified into unstructured knowledge sources and structured knowledge sources [42]. The detailed classification and examples are shown in Figure 1.

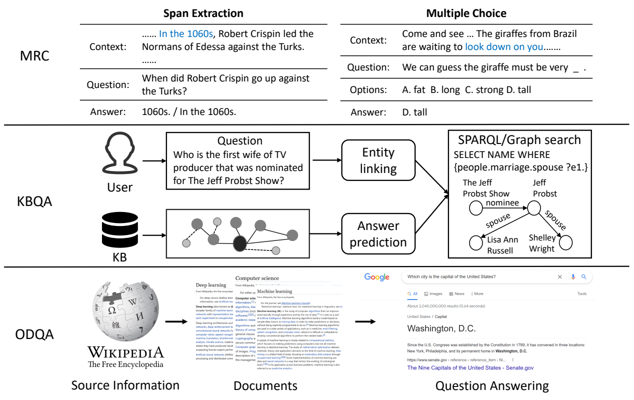
MRC
Span Extraction
Context: ...... In the 1060s, Robert Crispin led the Normans of Edessa against the Turks. ......
Question: When did Robert Crispin go up against the Turks?
Answer: 1060s. / In the 1060s.
Multiple Choice
Context: Come and see ... The giraffes from Brazil are waiting to look down on you.......
Question: We can guess the giraffe must be very _ .
Options: A. fat B. long C. strong D. tall
Answer: D. tall
KBQA
User
Question
Who is the first wife of TV producer that was nominated for The Jeff Probst Show?
KB
Entity linking
Answer prediction
SPARQL/Graph search
SELECT NAME WHERE
{people.marriage.spouse ?e1.}
The Jeff Probst Show
Jeff Probst
nominee
spouse
spouse
Lisa Ann Russell
Shelley Wright
ODQA
Wikipedia
The Free Encyclopedia
Source Information
Documents
Google
Which city is the capital of the United States?
Washington, D.C.
Question Answering

Figure 1: The general answer types and workflow of different QA systems. The example data are collected from open source datasets [47], [31], and [49].

For QA systems with unstructured knowledge sources, it can be further subdivided into systems that process a single document and cross-documents. The machine reading comprehension [57] is a typical type of QA system that processes a single document, i.e., given a question and a related single document, the system looks for answers from the context. This type of system is the most traditional QA system, which is widely used in question answering scenarios in commercial fields such as news and medical care [8, 43, 52]. Recently, deep learning algorithms are often applied to train language models to identify the themes and details of articles.

The systems that deal with cross-documents can be regarded as a type of information retrieval system [18], such as open-domain QA systems [11], which do not have question scopes, and generate answers by reasoning from multiple documents searched on the internet. Compared with the systems with a single document,

[1]https://github.com/SATE-Lab/QATest

this kind of system needs to identify and extract related multiple documents, and there are more steps such as logical reasoning.

For QA systems dealing with structured data, the mainstream technology is to answer a given question based on an already constructed knowledge base or knowledge graph [49]. The problem involves a variety of query types such as true and false judgments, quantitative statistics, etc. This type of system is mainly processed by entity name recognition, entity linking, and logical reasoning, and finally generates a query statement for querying the graph database, and outputs the query from the knowledge graph [25, 33, 49].

These QA systems receive different types of questions, and generate answers from different sources, which is one of the challenges of automated and systematic testing of QA systems. In this paper, we propose a uniform fuzzing framework that combines a series of metamorphic transformations with a test oracle to generate test data. Further, we design two testing criteria as guidance and a seed selection strategy in the fuzzing process to improve test efficiency.

3 APPROACH

In this section, we introduce the design and implementation of QATest, which achieves a uniform testing framework for QA systems. We first describe the design ideas and steps of the entire framework in Section 3.1, and then introduce the detailed designs of each module in Section 3.2, 3.3, and 3.4.

3.1 Overview of QATest

In traditional software testing, a coverage-guided fuzz testing framework usually maintains a test seed set and generates new test data based on mutation rules or other known interfaces. Then, after the test cases are executed, the framework decides whether to add the generated data to the seed set based on test criteria such as code coverage. The new test data is generated through continuous iteration and mutation to increase the test coverage.

Considering the effectiveness of fuzzing in traditional software testing, we propose a fuzzing framework for QA systems to detect erroneous behaviors. As depicted in Figure 2, the implementation of QATest is divided into three parts. First, a series of semantic transformations are constructed to generate new test data based on some seed data. Then, we propose two testing criteria to guide the test generation process and thus improve the bug detection efficiency. Finally, as the scale of the seed dataset increases, we further propose a seed selection strategy to diversify seeds and enhance the performance of QATest.

Specifically, Algorithm 1 presents the process of QATest generating test data for various types of QA systems. For a given seed test set S, we first design and implement a series of semantic transformations, which is introduced in Section 3.2. The set of all transformations is represented as T. Then, users can set the maximum number of transformation attempts *max_try*, and the number of seed samples *batch_size*, and QATest iteratively generates test cases based on these parameters. In each iteration, QATest selects a batch according to a seed sampling strategy proposed in Section 3.4 (Line 2). Next, QATest randomly transforms the data (Line 5), and judges whether the generated question meets the quality standard based on the ROUGE-1 score (Line 6). If the quality score is greater than the threshold, the generated question s' is added to the generated test set D (Line 7-9), otherwise, the next attempt is performed until the *max_try* is reached. Finally, based on the coverage testing criteria we designed and illustrated in Section 3.3, QATest evaluates which questions in the generated question batch can be sent back to the seed set, and updates the seed set (Line 10 - 11). After the iteration is complete, QATest outputs the final generated test set.

Algorithm 1: The overall process of QATest

```
Input: S: Seed test set, T: The set of transformations,
       batch_size: The batch size sampled in each iteration,
       max_try: The maximum number of trials
Output: D: The generated test set
1  $\tau \leftarrow$ setThreshold() ; // Threshold for evaluating the
     quality of the generated questions.
2  while batch_seed $\leftarrow$ pickSeedBatch(S, batch_size) do
3      batch_seed' $\leftarrow$ [];
4      for $s \in$ batch_seed do
5          $s' \leftarrow$ genTransData (T, s, max_try);
6          quality_score $\leftarrow$ Rouge1(s, s');
7          if quality_score $> \tau$ then // acceptable
8              D.append(s');
9              batch_seed'.append(s');
10     S.append (perStrategy (batch_seed', batch_seed));
11     S.append (gramStrategy (batch_seed', batch_seed));
12 return D
```

3.2 Test Generation with MRs

To simulate the possible question QA systems may receive in real usage scenarios, we implement three textual semantically consistent transformations, i.e., grammatical component-based, sentence structure-based, and adversarial perturbation-based transformations. Then, we employ a quality assessment method to discard low-quality generated questions.

3.2.1 Test Oracle.

The oracle problem is the most difficult task in software testing, i.e., the execution result of the program cannot be predicted. For QA systems, we cannot automatically obtain the correct answer for any input question, and the cost of manually labeling the answer is extremely high. Metamorphic testing is widely employed to generate tests for various kinds of complex software systems, such as autonomous driving systems [51], machine translation systems [24, 58], and dialogue systems [36].

Metamorphic Relations (MRs) refer to the relationship expected to be followed between input and output when the target program is executed multiple times. Specifically, we denote a seed question set Q, and the QA system as M that continuously receives natural language questions and outputs the corresponding answers. We define a series of semantically consistent transformations $\mathcal{T}$ for questions $q \in Q$. The generated question q' can be obtained by applying the transformation $t \in \mathcal{T}$ to the original question q, and q' has the same intent and meaning as q. Therefore, the QA system should output the same answer for the q and q'.

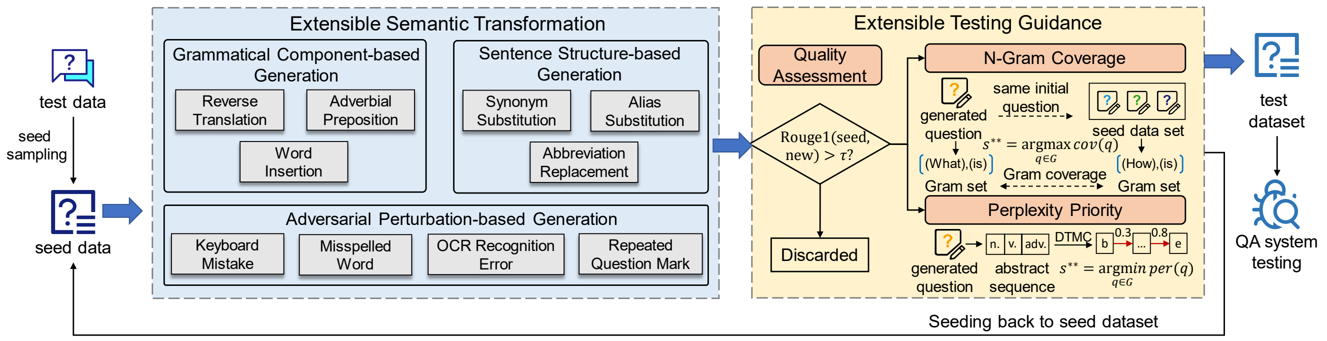

Figure 2: Overview of QATest.

Definition 3.1. Given a QA system M, a semantic transformation set $\mathcal{T}$, and a seed question set Q, a mutant question corresponding to $q \in Q$ with transformation $t \in \mathcal{T}$ can be represented as $q' = t(q)$. A mutant $q' \in \mathcal{T}(q)$ is regarded as a bug case of M if $M(q') \neq M(q)$.

3.2.2 *MR1: Grammatical Component-based Transformations.*

Grammatical component-based transformation generates new questions from the perspective of sentence transformation. As we all know, there are multiple expressions for a natural language sentence with the same meaning, so the operators we implement include the reverse translation of the question, i.e., the original question is translated into another language, and then translated back to English. Questions before and after translation have the same semantics, but there are likely to be minor differences in sentence structure or wording. The second operator is aimed at sentences with attributive phrases. In the original test set, most of the attributive phrases are posited at the end of the sentence. When we put it at the beginning of the sentence, the original meaning of the sentence is absolutely not changed. The last operator is to apply the most advanced pre-trained language model to insert a word or phrase into the original question. This operator can add some irrelevant words that do not affect the intent of the question under the premise of ensuring correct grammar.

3.2.3 *MR2: Sentence Structure-based Transformations.*

Sentence structure-based transformations pay more attention to some detailed vocabulary and structure in the question. For the non-entity words in the sentence, we implement the transformation through the synonym replacement operator. For entity words in sentences, we implement an operator for entity alias substitution. The last operator is some common abbreviations, such as questions starting with "what is" can be replaced by "what's" without changing the meaning of the question.

3.2.4 *MR3: Adversarial Perturbation-based Transformations.*

Adversarial perturbation-based transformation is designed to simulate some typos triggered by users when inputting questions. The first operator is the spelling mistakes caused by wrongly pressing similar keys on the keyboard. The second operator is the errors in OCR recognizing handwritten questions, such as recognizing "i" as "1". And the last two operators are word spelling mistakes, and repeated question marks at the end of the question. Examples of all transformations we have implemented are shown in Table 1.

3.2.5 *Quality Assessment.*

To ensure the quality of generated questions, we employ the ROUGE-1 metric [35] to evaluate the questions generated by metamorphic transformations. The metric ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics for evaluating automatic summarization and machine translation software in natural language processing. The ROUGE-1 score is computed by calculating the difference between the token set of each word in the candidate sentence and the reference sentence. Specifically, for a given original question q and the corresponding transformed question q', the ROUGE-1 score can be calculated as follows:

$$ROUGE-1 = \frac{\sum\limits_{s\in\{q\}} \sum\limits_{tok\in s} Count_{q'}(tok)}{\sum\limits_{s\in\{q\}} \sum\limits_{tok\in s} Count(tok)}$$

where tok is the set of word tokens in the question sentences.

3.3 Guidance Criteria Design

For conventional software programs, various structural code coverage is often employed as the guidance criteria in the fuzzing process. Test cases that can activate uncovered code are considered valuable data for testing the system and should be added to the seed set for further transformation. Different from traditional software, which consists of logical code, modern QA systems contain multiple deep learning algorithms and models, which rely on a large amount of labeled data to predict or classify new input data after training. The particularity of QA system construction makes the traditional code coverage method ineffective, which is confirmed in many studies of testing DNN models and DNN-based software [12, 21, 30, 38]. To overcome this challenge and ensure testing efficiency, we propose two testing criteria as guidance in the test generation process.

3.3.1 *N-Gram Coverage.*

N-Gram [10] is a classic statistical language model-based algorithm in NLP. Its basic idea is to perform a sliding window operation of size N on the content of sentences according to bytes, forming a sequence of byte fragments of length N. For the special circumstances when $N = 1$, 1-Gram is the set of each word in a given sentence. In this paper, we aim to evaluate the difference between the generated questions and the original seed set, and select the particular question that can increase the richness of the seed set

Table 1: Example questions with metamorphic transformations.

Transformations		Questions
Original question		What is a string over a Greek number when considering a computational problem?
MR1	Reverse translation	What is a string above a Greek number when considering a arithmetic problem?
	Adverbial preposition	When considering a computational problem, what is a string over a greek number?
	Word insertion	what role is a string over a Greek number when considering a computational problem?
MR2	Synonym substitution	What be a string over a Greek number when considering a computational problem?
	Alias substitution	What is a string over a Greek number when studying a computational problem?
	Abbreviation replacement	What's a string over a Greek number when considering a computational problem?
MR3	Keyboard mistake	What is a string over a Greek numbfr when considering a computational problem?
	Misspelled word	What is a string other a Greek number when considering a computational problem?
	OCR recognition error	What is a string ovek a Greek number when considering a computational problem?
	Repeated question mark	What is a string over a Greek number when considering a computational problem??

to improve the test generation efficiency. Therefore, we apply N-grams to represent the generated questions, thereby evaluating the differences between the generated questions compared to the original seed set.

For QATest, the seed set is updated dynamically, and the newly added data is generated by at least one transformation of the original seed data. Assuming that the generated question q is based on s_0 in the original seed set S with one or more iterative transformations, our goal is to calculate the differences of N-Gram corresponding to q and other questions transformed from s_0 in S, which can be regarded as homologous data. The differences between the newly generated data and its homologous data are considered to decide whether it should be added to the seed set. The definition of homologous data is described as follows:

Definition 3.2. The data in the seed set with the same original seed is considered as homologous data. Suppose S is the seed data set and q is the newly generated data, the homologous data of q is represented as:

$$H(q) = \{s | s \neq q \wedge ori(s) = ori(q) \wedge s \in S\}$$

The generated questions that can stimulate large differences are considered to be more likely to generate new sentence patterns and trigger potential erroneous behaviors in QA systems. Therefore, the N-Gram coverage is calculated as follows:

$$cov(q) = \frac{\sum_k |\text{k-Gram}(q) - \bigcup_{s \in H(q)} \{\text{k-Gram}(s)\}|}{\sum_k | \bigcup_{s \in H(q)} \{\text{k-Gram}(s)\}|}$$

where k is set as $k = \{1, 2, 3, 4\}$ in this paper, because the transformation applied by QATest generally results in few changes in a question, an excessively large N-Gram may cause the data-sparse problem, leading to the ineffectiveness of the criteria.

Finally, the N-Gram coverage criterion as guidance for selecting data to feedback to the seed set is defined as follows:

Definition 3.3 (N-Gram coverage). For each fuzzing iteration, the generated data batch is donated as G and each generated question is represented as $q \in G$. The question selected for feedback to the seed set is regarded as:

$$s^* = \arg\max_{q \in G} cov(q)$$

3.3.2 *Perplexity Priority.*

In the field of natural language generation, perplexity [26] is usually applied to calculate the authenticity of the generated sentences, so as to evaluate the quality of language models [10]. Generally, the probability of a sentence reflects the perplexity and is calculated based on Markov theory [26], which regards the probability of each word in the question only related to a limited one or a few words that appear before it. Thus, perplexity can be represented by the product of the probabilities of adjacent N-Grams in a sentence.

In this paper, we refer to the traditional evaluation criteria of perplexity and design a perplexity priority criterion as seed data feedback guidance. Since QATest samples seed data and applies transformations to generate questions in each iteration, the authenticity of the generated questions may decrease as the iteration batch increases. To ensure that the generated test questions have good authenticity and rationality, we consider applying perplexity priority as guidance for seed data retention. Different from the traditional perplexity of calculating the 1-Gram of the generated question directly, we first identify the part of speech of each word and construct the corresponding abstract sequence for calculation. Because the 1-Gram of the sentence before and after the transformation is very similar, the part-of-speech changes can better reflect small changes in sentence patterns. The abstract sequence is defined as follows:

Definition 3.4 (Abstract sequence). For the part-of-speech list L, $l \in L$ represents each specific part of speech, and b and e represent the start and end states of the question, respectively. Given a generated question with n words, the corresponding abstract part-of-speech sequence s can be denoted as:

$$s = seq(q) = bl_1 l_2 \dots l_n e$$

where l_i is the part-of-speech of i-th word in the generated question.

The abstract part-of-speech sequence of input questions can be considered as a discrete-time Markov chain [44], i.e., the probability of the part-of-speech of the current token in the sentence depends only on the previous token. Therefore, based on the abstract sequence, the probability of a given generated question can be calculated as follows:

$$p(s) = p(b)p(t_1|b)p(t_2|t_1) \dots p(e|t_n), \text{where } p(b) = 1$$

According to the calculation of conditional probability, for the part-of-speech $a, b \in L$ corresponding to two adjacent words and the current seed set S, $p(a|b)$ can be calculated as follows:

$$p(a|b) = \frac{|(b, a) \text{ in 2-Gram}(S)|}{|(\_, a) \text{ in 2-Gram}(S)|}$$

where 2-Gram(S) is the set of 2-Gram pairs of the seed data set, and $(\_, a)$ represents all 2-Gram pairs ending with a. Note that S is updated in each iteration and 2-Gram(S) is refreshed accordingly. Therefore, the probability of two part-of-speech is different in each iteration and represents the probability in the current state of the seed data. Finally, similar to the traditional perplexity computation, the perplexity of the abstract sequence $seq(q)$ can be represented as:

$$per(q) = p(seq(q))^{-1/n}$$

In each iteration, `QATest` selects the generated question with the lowest perplexity score in a generated batch. The perplexity priority for guiding seed data feedback is defined as follows:

Definition 3.5 (Perplexity priority). For each fuzzing iteration, the generated data batch is donated as G and each generated question is represented as $q \in G$. The question selected for feeding back to the seed set is regarded as:

$$s^{**} = \arg\min_{q \in G} per(q)$$

3.4 Seed Selection Strategy

As the number of iterations increases, the scale of the seed set increases, and it is necessary to apply some seed data selection strategies to sample seed data for transformations. In this paper, we record the original seed data corresponding to each question in the seed set, then each original seed data corresponds to a number, which represents how many of its variants are in the seed set. To ensure the diversity of tests and generate more test cases with different sources, we prefer to select seeds with few variants for further transformation and generation.

Specifically, `QATest` traverses the seed queue and determines whether the current seed is selected according to its probability. It imposes a bias by assigning a high probability to the original seed with few variants and its corresponding transformed data in the seed queue. The probability of selecting a seed s is computed by:

$$P(s) = 1/g(ori(s))$$

where $ori(s)$ represents the initial seed corresponding to the seed question s, and $g(s_0)$ is the num of seeds that are transformed by the initial seed s_0. Note that after each iteration of updating the seed set, the value of $g(s_0)$ is updated accordingly.

Along with the seed selection strategy, `QATest` can guarantee the diversity of the generated tests, because it performs similar multiple iterations of transforms on all initial cases, avoiding too many transformations for a small number of original seeds. Besides, since the initial data in the seed set can not be eliminated, some questions with more mutations still have the opportunity to be selected to generate new test cases, which can increase the cases triggering potential bugs in the QA systems.

4 EXPERIMENT DESIGN

In this section, we introduce the experiment setup, including research questions, QA datasets and systems under test, and the implementation of question transformations. To conduct the experiments, we implement `QATest` upon Python 3.6 [2] and Pytorch 1.8.0 [3]. All the tested QA systems are reproduced according to the parameters given in the paper and constructed on a Ubuntu 18.04.3 LTS server with Tesla T4 and 15GB graphics memory.

4.1 Research Questions

`QATest` is designed to systematically test various types of QA systems with different input question types. To this end, we empirically explore the following four research questions (RQ):

RQ1 (Metamorphic Transformations): *How effective are different transformation methods for generating tests for various types of QA systems?* In this RQ, we aim to provide a global picture of the effectiveness of `QATest` in revealing the erroneous behaviors of QA systems. We separately show the test case generation effect of three types of MRs proposed in this paper, and compare them with other existing QA systems testing techniques.

RQ2 (Testing Criteria): *How effective are the different seed guidance criteria for detecting erroneous behaviors of QA systems?* To evaluate the effectiveness of our proposed testing criteria as test generation guidance, we separately verify the effectiveness of generating test data with one guidance and without any guidance, respectively.

RQ3 (Test Diversity): *How diverse are the bug-triggering tests generated by different transformation methods and strategies?* Referring to the diversity metrics proposed by other fuzzing testing frameworks for DNN models [53], we verify the diversity of failed tests generated by each MR and guidance, and compare the diversity with existing QA system testing techniques.

RQ4 (Case Study): *Analysis of the bug types detected by* `QATest`. As the first uniform fuzzing framework for various types of QA systems, we analyze and classify the erroneous behaviors found by `QATest`, which could be useful for the optimization of QA systems in future research.

4.2 Datasets and QA Systems

4.2.1 Datasets. As shown in Table 2, we experiment `QATest` with six popular publicly available datasets, including SQuAD 1.1 [48], SQuAD 2.0 [47], BoolQ [17], Race [31], WebQuestions (WebQ for short) [9], and CQA [49]. According to the form of answers, we choose CQA-Verification (CQA-Ver. for short) and CQA-Comparative (CQA-Com. for short) in the CQA dataset.

Table 2: The details of Question Answering Datasets.

Dataset	QA types	Answers	# Tests	Description
SQuAD 1.1	MRC	NLP	10,570	Questions collected from Wiki articles
SQuAD 2.0	MRC	NLP/NoAns	11,873	
BoolQ	MRC	YES/NO	3,270	Boolean questions
Race	MRC	A/B/C/D	11,873	From English tests
WebQ	KBQA	NLP	2,032	Google Suggest API
CQA-Ver.	KBQA	YES/NO	10,150	Subset of complex sequential QA data
CQA-Com.	KBQA	NLP	7,902	

Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowd workers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage. SQuAD 1.1 is the previous version of the SQuAD dataset, which is constructed on 500+ articles. SQuAD2.0 combines the 100, 000 questions in SQuAD 1.1 with over 50, 000 unanswerable questions written adversarially by crowd workers to look similar to answerable ones.

BoolQ is a boolean question answering dataset released by Google research. The data in BoolQ are naturally occurring and generated in unprompted and unconstrained settings.

Race dataset is collected from the reading comprehension of middle school and high school English exams, which contains 28, 000 short passages and nearly 100, 000 questions. Each piece of context corresponds to multiple question-and-answer pairs, and each question is a multiple-choice question with at least four options.

WebQuestions dataset is built by Stanford researchers using the Google Suggest API. WebQuestions dataset is designed to assist in the construction of knowledge graphs and KBQA systems, which contains various common-sense question-and-answer pairs.

CQA dataset is a question-and-answer dataset constructed based on the wikidata knowledge graph. It contains 6 question types, i.e., simple, logical, quantitative, comparative, verification, quantitative count, and comparative count questions. We mainly choose verification and comparative in our experiments. The verification type can be regarded as boolean questions and the answer type of comparative questions is natural language.

4.2.2 QA systems. To evaluate the performance of `QATest`, we select four state-of-art QA systems as the testing subjects in our experiments. We briefly introduce them as follows:

UnifiedQA [29] is a QA platform that converts a variety of different question formats into a unified format and outputs answers. UnifiedQA provides the state-of-the-art pre-trained language model T5 (Text-To-Text Transfer Transformer) [46] and corresponding versions of different sizes. We apply the T5-large in our experiments.

ALBERT [34], a "lite" version of BERT proposed by Google, is a popular unsupervised language representation learning algorithm. It can be applied to multiple NLP tasks, including QA task, and has achieved high scores on the Leaderboard of Squad 2.0 [4].

DrQA [11] is a system for reading comprehension applied to open-domain question answering, which is released by FaceBook. It employs Wikipedia as the knowledge source and generates answers by finding the relevant documents from large-scale source documents and identifying the answers from those documents.

MARL [25] is a KBQA system that answers factual questions through complex inferring over a realistic-sized KG of millions of entities. We employ MARL as a KBQA system for testing.

4.3 Sentence Transformations

As described in Section 3.2, `QATest` generates tests by making realistic transformations on the original data with ten transformations in total. Most of the transformations are implemented with a tool named nlpaug [37], which is a python library that helps users with augmenting NLP for machine learning projects.

Specifically, we employ the WordNet library [39] to identify the synonym set for a given word and achieve the synonym replacement transformation. For adverbial preposition transformation, we mainly identify the when-guided time adverbial and the if-guided conditional adverbial in questions. For inserting word transformation, the most commonly used pre-trained language model BERT [20] is applied to assist in inserting words or phrases. The back translation operator also employs the pre-trained language model to first translate the English original question into German and then translate it back to English. For the entity alias substitution operator, we apply one of the best entity linking tools in science, i.e., TagMe [5], to help us identify and replace entities.

Note that for each transformation, if the generated question is exactly the same as the original question, it is considered a failed transformation, and the generated question will not be retained. Similarly, if the transformation requirements can not be satisfied, the transformation will not be performed. For example, when there is no adverbial clause in the original question, the transformation of the adverbial preposition will not be successfully applied.

4.4 Baseline Approaches

To evaluate the effectiveness of `QATest`, we compare it with two existing QA system testing methods, i.e., MT4MRC [14] and QAAskeR [13]. Specifically, the metamorphic relationships proposed by MT4MRC can only be established on the dataset whose answer is YES/NO, so only the tasks corresponding to the BoolQ dataset can be applied. QAAskeR generates new questions after converting given question and answer pairs into declarative sentences, including three transformations: (1) generating new wh-questions based on wh-questions; (2) generating new general questions based on wh-questions; and (3) generating new wh-questions based on general questions. If there is no general question in the data set, the corresponding transformation relationship fails to be generated.

5 RESULT ANALYSIS

This section presents the experiment result and then analyzes the performance of our approach.

5.1 Answer to RQ1: Metamorphic Transformations

To evaluate the effectiveness of each type of MR applied by `QATest`, we select 500 test cases from each data set and apply the MRs proposed in Section 3.2 to generate test data sets. Then, we record the number of failed tests in the generated test set on the corresponding QA systems, which is shown in Table 3. In addition, we also show the number of failed tests corresponding to applying three MRs together, the baseline methods, and `QATest` framework in the table.

For all QA systems, applying the three types of MR alone can generate more failed tests than the original test set, which indicates that each type of MR has the ability to reveal erroneous behaviors and discover more potential flaws in QA systems than the original test set. Further, compared with the direct application of MR to generate test cases, `QATest`, as a uniform fuzzing framework, can generate a larger set of failed tests based on the same seed set, which implies that the effectiveness of the test generation guidance and the seed selection strategy employed by `QATest`.

Table 3: The number of failed tests generated by different approaches with 500 original test cases as the seed dataset.

	System	Dataset	Ori.	MR1	MR2	MR3	All	QAAskeR	QATest
MRC	UnifiedQA	SQuAD 2.0	81	214	159	363	736	131	**3,812**
		BoolQ	78	270	157	351	778	909[2]	**4,228**
	ALBERT	SQuAD 2.0	102	236	201	496	940	141	**4,711**
		Race	169	303	244	624	1,171	-	**5,335**
ODQA	DrQA	SQuAD 1.1	405	608	477	1,554	2,639	320	**12,207**
		WebQ	417	629	637	1,600	2,866	358	**12,333**
KBQA	MARL	CQA-Ver	78	71	103	296	469	-	**1,895**
		CQA-Com	289	521	432	1,158	2,101	-	**8,271**

Compared with existing QA system testing methods, our method has better bug detection ability, and based on the same original test set, `QATest` can generate more failed tests. The MT4MRC method generates test cases based on a set of MRs. Because it can perform transformations with reverse answers, its ability to find erroneous behaviors is slightly better than directly applying all MRs. However, after the test cases are transformed iteratively in `QATest` framework, the ability to detect defects is gradually enhanced, so the number of failed tests generated by `QATest` is more than that of MT4MRC. The ability of QAAskeR to discover defective behaviors is a little limited, mainly because it can only generate one or even fail to generate a corresponding new test case for a given original test data. Moreover, `QATest` has better universality compared with other existing QA system testing methods. It can generate test cases for questions corresponding to various question types, and can be commonly used in MRC, ODQA, and KBQA systems because `QATest` is less restrictive for question formats.

5.2 Answer to RQ2: Guidance Criteria

To evaluate the effectiveness of our proposed guidance criteria, we applied `QATest` to verify the effects of applying both criteria and applying two criteria simultaneously to guide the test generation, and compared with those without any coverage guidance. The number of failed tests generated by these four strategies on each tested system and the dataset is shown in Table 4, where the cells with gray background represent the corresponding failed test rates. As can be discovered from the table, the test sets generated using the two testing criteria as guidance can generate more failed tests and achieve higher bug detection rates than those generated without guidance. This indicates that our proposed coverage criteria are effective to improve test efficiency and can generate more test cases that can trigger erroneous behaviors of QA systems.

Figure 3 shows the number of failed tests generated by each testing criteria guidance on the UnifiedQA system with the number of fuzzing iterations. The gray horizontal line represents the number of failed tests generated by baseline methods. Because MT4MRC and QAAskeR cannot be iteratively generated multiple times, their corresponding failed test numbers are fixed values. As can be seen from Figure 3, as the iteration number increases, the

[2]This is the number of failed tests generated by MT4MRC [14]. QAAskeR cannot generate new questions for YES/NO questions. In addition, MT4MRC can only be applied on YES/NO questions.

Table 4: The number and proportion of failed tests in the test set generated by different strategies with 3,000 iterations.

System	Dataset	No guidance		Gram		Per.		Gram+Per.	
UnifiedQA	SQuAD 2.0	2,953	20.5%	**3,853**	**27.2%**	3,696	25.9%	3,812	27.0%
	BoolQ	3,072	21.3%	4,064	28.4%	3,805	26.7%	**4,228**	**29.8%**
ALBERT	SQuAD 2.0	3,875	26.9%	4,606	**32.6%**	4,688	32.8%	**4,711**	33.4%
	Race	5,098	35.0%	5,328	**37.2%**	5,171	36.0%	**5,335**	**37.2%**
DrQA	SQuAD 1.1	11,948	85.3%	12,096	87.6%	12,186	88.0%	**12,207**	**88.9%**
	WebQ	12,000	87.5%	12,275	90.3%	**12,362**	90.8%	12,333	**91.1%**
MARL	CQA-Ver	1,840	14.9%	1,906	15.5%	**1,951**	**16.1%**	1,895	15.6%
	CQA-Com	**8,425**	58.2%	8,363	**58.4%**	8,317	57.6%	8,271	57.6%

method without guidance is less effective than `QATest`. The guidance effect based on perplexity priority is slightly lower than the guidance method based on N-Gram coverage, because the perplexity priority is designed to ensure the authenticity of the seed set, and the change of text content is relatively small compared to the original question. Further, since the existing QA system testing methods only generate test cases based on some MRs, the number of generated failed tests is limited, and it is difficult to ensure that as many defect behaviors of the QA system can be found as possible.

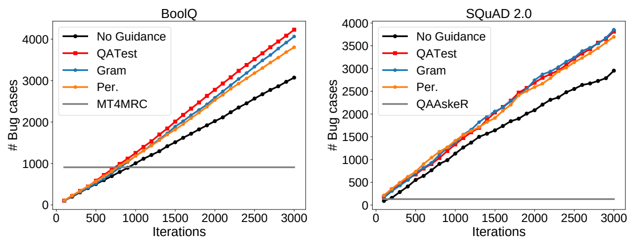

Figure 3: The number of failed tests on the UnifiedQA system for the test set generated with different test generation criteria at each iteration step.

5.3 Answer to RQ3: Test Diversity

In RQ3, we try to analyze the diversity of bug-triggering tests generated by `QATest`, i.e., whether all the original test cases can detect the erroneous behaviors of the QA systems. Therefore, we count the number of different original seed data corresponding to the failed tests generated by each method. To reflect the diversity of bug categories, we calculate the ratio of the number of seeds that can trigger bugs to the total number, which is expressed as diversity. The metric of diversity is referred to as a testing framework for DNNs [53]. The experiment results are shown in Table 5.

From Table 5, we can observe that `QATest` with both N-Gram coverage and perplexity priority testing criterion as test generation guidance can achieve the highest diversity in most circumstances. For the UnifiedQA, ALBERT, and DrQA systems, applying both N-Gram coverage and perplexity priority as guidance for updating seed data has higher diversity than using a single guidance method. For the MARL system, the diversity of applying different strategies is similar. The main reason is that in the CQA dataset, the entity and relation triples of the knowledge graph corresponding to many questions are similar, and it is difficult to generate test cases for each entity relation.

Table 5: The test diversity found by `QATest` with different metamorphic relations and test generation guidance.

System	Dataset	MR1	MR2	MR3	ALL	QAAskeR	QATest (Gram)	QATest (Per.)	QATest
UnifiedQA (MRC)	SQuAD 2.0	33.80%	23.40%	29.60%	46.40%	26.20%	68.20%	69.00%	**71.20%**
	BoolQ	41.60%	22.60%	29.40%	52.80%	80.40%[2]	82.00%	83.80%	**85.40%**
ALBERT (MRC)	SQuAD 2.0	55.20%	34.20%	29.00%	43.60%	28.20%	74.20%	76.60%	**79.60%**
	Race	43.40%	36.60%	53.00%	59.80%	-	87%	86.40%	**89%**
DrQA (ODQA)	SQuAD 1.1	78.00%	71.20%	93%	96.20%	64%	97.80%	97.80%	**98%**
	WebQ	90.40%	85.80%	96.60%	98.60%	71.60%	99.80%	99.80%	**100%**
MARL (KBQA)	CQA-Ver	13.60%	15.00%	16.20%	16.20%	-	**16.60%**	**16.60%**	16.40%
	CQA-Com	57.80%	58.00%	58.60%	**59.20%**	-	58.80%	**59.20%**	58.80%

`QATest` can achieve high diversity mainly because multiple iterations can generate more test cases that trigger erroneous behaviors of QA systems. For the test set generated with guidance criteria and seed selection strategy, Figure 4 shows the number of test cases corresponding to different iterations, and the proportion of corresponding failed tests. The number of test cases generated through one iteration is the largest, and the most iterations in the test set are 11 times, with only 1 test case. However, among the test cases generated after one round of iteration, the proportion of failed tests is relatively small. Therefore, the fuzzing process with the seed selection strategy can effectively improve the test efficiency, trigger more potential erroneous behaviors and improve test diversity.

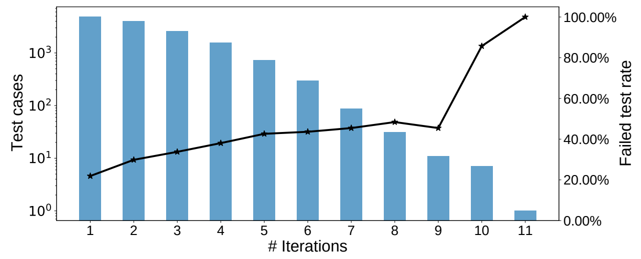

Figure 4: The number of transformations applied on each test case and the corresponding failed test rate on UnifiedQA system and BoolQ dataset.

5.4 Answer to RQ4: Case Study

In this RQ, we investigate and classify the erroneous behaviors of the QA system detected by `QATest`. In Table 6, we show five types of representative erroneous behaviors. We list the questions and expected answers in the original test set, as well as the new questions generated by `QATest` and the actual answers of the system, and the corresponding intercepted context corresponding to the questions. Specifically, we summarize the bug types below.

(1) <NoAnswer> questions are answered with irrelevant content. As shown in example 1, the SQuAD 2.0 dataset contains many unanswerable questions, but the questions are related to the original context. One of the bug types is that the QA system cannot identify whether the question can be answered or not, and output a keyword from the original text as the answer.

(2) Answerable questions are answered with <NoAnswer>. Contrary to example 1, another common misbehavior is that the system returns <NoAnswer> for a question that could be answered. As shown in example 2, the question has a corresponding answer in the context, but the system can't find a matching answer and outputs <NoAnswer>.

(3) Answers that are unrelated to the question. Another type of error is that the QA system's answers are not related to the question. In example 3, the question is asking about the year, but the output of the QA system is a person's name.

(4) Answers with wrong keywords in the context. Example 4 shows a relatively rare mistake, i.e., the keyword position of the system's answer in the context is very close to the correct answer, but the system extracts the improper keyword as the answer.

(5) Answers without logical reasoning. An advanced QA system should be able to deal with some logical reasoning because some questions do not have a direct correspondence to the context. However, in example 5, the QA system triggers a bug by outputting a keyword in the original sentence without logical reasoning.

6 DISCUSSION

This section discusses the comparison with existing testing approaches, the effectiveness of `QATest`, and threats to validity.

6.1 Comparison with Existing Approaches

By comparing `QATest` with the other metamorphic transformation-based test data generation approaches, we demonstrate that `QATest` is more effective and general-applied than existing methods. First of all, unlike other existing methods for generating test cases based on MRs with limitations, our proposed fuzzing framework has a wider range of application scenarios. As the experiment results of RQ1 and RQ3 show, MT4MRC and QAAskeR cannot generate test cases for some question and QA system types. Moreover, another benefit of `QATest` is that we can handle situations where there are multiple correct answers corresponding to a question. The existing automated testing method QAAskeR converts known questions and answers into declarative sentences and then generates new questions, but cannot guarantee that the correct answer is unique.

As a general QA system testing framework, `QATest` can automate the sampling of seed data, the generation of new test data, and updating seed data based on testing criteria, which can improve

Table 6: The examples of erroneous behaviors detected by QATest on ALBERT QA system.

Example		Original/Transformed Questions	Expected/Actual Answers
#1	Original	Q: What is the most common form of Norman art in churches?	A: <NoAnswer>
	Transformed	Q: What is the most commom form of Norman art in churches?	A: stonework or metalwork
	Context	... In Britain, Norman art primarily survives as stonework or metalwork. such as capitals and baptismal fonts. ...	
#2	Original	Q: What were the origins of the Raoulii family?	A: an Italo-Norman named Raoul
	Transformed	Q: What were the probable origins of the Raoulii family??	A: <NoAnswer>
	Context	... The Raoulii were descended from an Italo-Norman named Raoul, the Petraliphae were descended from a Pierre d'Aulps ...	
#3	Original	Q: What year did Roger de Tosny fail to accomplish what he set out to do?	A: 1018
	Transformed	Q: In what year did Roger de Tosny not achieve what he had planned??	A: Roger de Tosny
	Context	... In 1018, Roger de Tosny travelled to the Iberian Peninsula to carve out a state for himself from Moorish lands, but failed ...	
#4	Original	Q: What architecture type came before Norman in England?	A: Anglo-Saxon
	Transformed	Q: What type of architecture came before Norman in England?	A: Early Gothic
	Context	... the period of Norman architecture immediately succeeds that of the Anglo-Saxon and precedes the Early Gothic ...	
#5	Original	Q: What is a commonly used measurement used to determine the complexity of a computational problem?	A: time
	Transformed	Q: What's a common measuring instrument to determine the complexity of a computer problem?	A: algorithm
	Context	... Thus the time required to solve a problem (...) is calculated as a function of the size of the instance. In particular, ...	

the diversity of error types and detect a large number of erroneous behaviors of QA systems. The results from RQ3 show that our proposed two guidance criteria can effectively guide the test generation and generate more failed tests compared to the method without guidance. For various types of QA systems, after model training and development, QATest can be employed to automatically generate large-scale test data to verify the robustness of the developed system and reduce the cost of manual data annotation and testing.

6.2 The Effectiveness of QATest

QATest is a fuzzing framework for mainstream QA systems that combines semantically consistent data transformation, quality assessment, guidance testing criteria, and seed selection strategies. From the results of RQ1, each MR implemented by QATest is effective, i.e., the generated test set can detect more defects than the original test set. And after applying the guidance criteria and seed selection strategies, the number of detected defects has increased substantially. Similarly, from the results of RQ2 and RQ3, after applying our proposed testing guidance and iterative fuzzing framework, the test diversity of the generated test set is improved compared to directly applying MRs for test generation. We consider that the guidance testing criteria proposed in this paper play an important role in the effectiveness of QATest. On the one hand, our proposed N-Gram coverage can enrich the diversity of question sentences in the seed set. For example, variations in phrases and phrases imply a variety of sentence patterns. On the other hand, the perplexity priority can assist in ensuring the authenticity of the seed set and maintaining the real meaning of the generated questions after iterative transformations.

6.3 Threats to Validity

Test subject and dataset selection. The selection of QA systems and datasets is one of the primary threats to validity. With the rapid development of QA systems, there are more and more types and application scenarios for QA systems. Each system is constructed differently and based on variant DNN models, and the effectiveness is also different. We alleviate this threat by employing four commonly used and well-known QA systems, which are categorized as reading comprehension, knowledge graph, and open-domain QA systems. Besides, we conduct experiments on six open-source and widely-used QA datasets to avoid the potential impact of datasets.
Data transformation implementation. The last threat comes from the implementation of transformation operators. Although the transformed operators are well-designed and are commonly seen in the actual environment, it is impossible to guarantee the reliability of the generated question without manual inspection. To ensure the quality of the generated questions, most of the transformations are implemented with pre-trained language models such as the standard WordNet library and BERT. Besides, we apply the existing ROUGE metric to evaluate the quality of the generated data and discard the low-quality generated questions directly.

7 RELATED WORK

In this section, we compare our work with other testing approaches for QA systems and other deep learning-based systems. We primarily focus on two perspectives: (1) the testing of QA systems; and (2) the fuzzing for deep learning-based systems.

7.1 The Testing of QA Systems

There are two existing metamorphic testing approaches for QA systems proposed by Chen et al. [13, 14]. Specifically, MT4MRC [14] proposed a series of MR relationships for YES/NO answer types and generated test cases for QA system testing based on reading comprehension. QAAskeR [13] converted the given question and answer pairs into declarative sentences, and then generated new questions for recursively input into the QA system and detected erroneous behaviors. Different from these existing QA system testing methods, our proposed QATest can perform general testing on QA systems without the limitation of system type or answer type. In addition, based on our proposed test criteria applied to guide test generation, a large scale of test cases that trigger system errors can be generated, and QA systems can be more fully tested.

On the other hand, there are some techniques for building test datasets. Gu et al. [23] built a test benchmark based on the data distribution to evaluate the generalization of KBQA systems. Another work [41] referred to the construction method of the SQuAD dataset [47] and constructed a dataset on other data sources, which can be used to verify the robustness of QA systems. Different from the above test set construction methods, which rely on a large number of crowdsourced labeled data, our proposed QATest framework has a higher degree of automation, and only requires the original test set to generate tests for QA systems.

7.2 The Fuzzing for DL Systems

The fuzzing technique has been widely applied in many deep learning-based systems to identify erroneous behaviors. TensorFuzz [45] was a coverage-based fuzzing framework that generated variants through multiple iterations to trigger the misbehavior of neural networks, and discovered some real bugs in DNN models. To detect the bugs of operators dealing with high-dimensional tensors in the DL framework, DUO [56], a differential fuzzing framework, was proposed to combine fuzzing techniques and differential testing to identify the erroneous outputs of DL operators. Besides, Shen et al. [50] proposed another fuzzing method for testing DL operators, which took inconsistency as feedback to guide fuzzing and can detect the NaN bugs and crashes. DeepHunter [53] was a coverage-guided fuzzing framework for deep neural networks, which implemented a series of data mutation and seed selection strategies based on some existing testing criteria for DNNs.

Different from the fuzzing framework mentioned above, our proposed approach is a general testing framework for QA systems, rather than for a certain type of DNN models or operators. Moreover, we design two testing criteria, especially for question-answer data type to guide the seed generation in fuzzing, which can effectively improve the test efficiency and the diversity of generated tests.

8 CONCLUSION

In this paper, we proposed and evaluated QATest, a uniform fuzzing framework for QA systems. To enhance the abundance of tests and detect more erroneous behaviors of QA systems, we implement three kinds of metamorphic transformations to generate test questions. Further, we propose two testing criteria, i.e., the N-Gram coverage and the perplexity priority, to guide the test generation process and improve test diversity and efficiency. QATest has been evaluated on four state-of-art QA systems and six widely-used datasets. The experiment results show that the transformations applied by QATest can detect a large number of erroneous behaviors of QA systems. In addition, the proposed guidance testing criteria can be employed to increase testing efficiency and diversity. In the case study, we analyzed the bug types of QA systems, and we believe that QATest provides a foundation for the development of testing and analysis of QA systems.

ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their constructive comments. This project was partially funded by the National Key R&D Program of China (2021ZD0112903) and the National Natural Science Foundation of China under Grant No. 62002158.

REFERENCES

[1] [n.d.]. Amazon promises fix for creepy Alexa laugh - BBC News. https://www.bbc.com/news/technology-43325230. (Accessed on 05/05/2022).

[2] [n.d.]. Python Release Python 3.6.0 | Python.org. https://www.python.org/downloads/release/python-360/. (Accessed on 05/05/2022).

[3] [n.d.]. PyTorch. https://pytorch.org/. (Accessed on 05/05/2022).

[4] [n.d.]. The Stanford Question Answering Dataset. https://rajpurkar.github.io/SQuAD-explorer/. (Accessed on 05/07/2022).

[5] [n.d.]. TagMe - TagMe API. https://services.d4science.org/web/tagme/tagme-help. (Accessed on 04/30/2022).

[6] [n.d.]. Tay: Microsoft issues apology over racist chatbot fiasco - BBC News. https://www.bbc.com/news/technology-35902104. (Accessed on 05/05/2022).

[7] Razieh Baradaran, Razieh Ghiasi, and Hossein Amirkhani. 2020. A survey on machine reading comprehension systems. *Natural Language Engineering* (2020), 1–50.

[8] Asma Ben Abacha and Dina Demner-Fushman. 2019. A Question-Entailment Approach to Question Answering. *BMC Bioinform.* 20, 1 (2019), 511:1–511:23. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-019-3119-4

[9] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, USA, 1533–1544. https://aclanthology.org/D13-1160

[10] William B Cavnar, John M Trenkle, et al. 1994. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, Vol. 161175. Citeseer.

[11] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051* (2017).

[12] Junjie Chen, Ming Yan, Zan Wang, Yuning Kang, and Zhuo Wu. 2020. Deep neural network test coverage: How far are we? *arXiv preprint arXiv:2010.04946* (2020).

[13] Songqiang Chen, Shuo Jin, and Xiaoyuan Xie. 2021. Testing Your Question Answering Software via Asking Recursively. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 104–116. https://doi.org/10.1109/ASE51524.2021.9678670

[14] Songqiang Chen, Shuo Jin, and Xiaoyuan Xie. 2021. Validation on Machine Reading Comprehension Software without Annotated Labels: A Property-Based Method. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 590–602. https://doi.org/10.1145/3468264.3468569

[15] KR1442 Chowdhary. 2020. Natural language processing. *Fundamentals of artificial intelligence* (2020), 603–649.

[16] Philipp Cimiano, Christina Unger, and John McCrae. 2014. Ontology-based interpretation of natural language. *Synthesis Lectures on Human Language Technologies* 7, 2 (2014), 1–178.

[17] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *NAACL*.

[18] Nicola De Cao, Wilker Aziz, and Ivan Titov. 2018. Question answering by reasoning across documents with graph convolutional networks. *arXiv preprint arXiv:1808.09920* (2018).

[19] Jan Deriu, Alvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. 2020. Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review* (2020), 1–56. https://doi.org/10.1007/s10462-020-09866-x

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[21] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. DeepGini: Prioritizing Massive Tests to Enhance the Robustness of Deep Neural Networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual Event, USA) *(ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 177–188. https://doi.org/10.1145/3395363.3397357

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[23] Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: Three Levels of Generalization for Question Answering on Knowledge Bases. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3477–3488. https://doi.org/10.1145/3442381.3449992

[24] Pinjia He, Clara Meister, and Zhendong Su. 2020. Structure-Invariant Testing for Machine Translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 961–973. https://doi.org/10.1145/3377811.3380339

[25] Yuncheng Hua, Yuan-Fang Li, Gholamreza Haffari, Guilin Qi, and Wei Wu. 2020. Retrieve, Program, Repeat: Complex Knowledge Base Question Answering via Alternate Meta-learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, Christian Bessiere (Ed.). ijcai.org, 3679–3686. https://doi.org/10.24963/ijcai.2020/509

[26] Dan Jurafsky. 2000. *Speech & language processing.* Pearson Education India.

[27] Veton Kepuska and Gamal Bohouta. 2018. Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 99–103. https://doi.org/10.1109/CCWC.2018.8301638

[28] Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring compositional generalization: A comprehensive method on realistic data. *arXiv preprint arXiv:1912.09713* (2019).

[29] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. Unifiedqa: Crossing format boundaries with a single qa system. *arXiv preprint arXiv:2005.00700* (2020).

[30] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[31] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683* (2017).

[32] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Complex Knowledge Base Question Answering: A Survey. *arXiv preprint arXiv:2108.06688* (2021).

[33] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. A survey on complex knowledge base question answering: Methods, challenges and solutions. *arXiv preprint arXiv:2105.11644* (2021).

[34] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[35] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[36] Zixi Liu, Yang Feng, and Zhenyu Chen. 2021. *DialTest: Automated Testing for Recurrent-Neural-Network-Driven Dialogue Systems.* Association for Computing Machinery, New York, NY, USA, 115–126. https://doi.org/10.1145/3460319.3464829

[37] Edward Ma. 2019. NLP Augmentation. https://github.com/makcedward/nlpaug.

[38] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. *DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems.* Association for Computing Machinery, New York, NY, USA, 120–131. https://doi.org/10.1145/3238147.3238202

[39] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[40] John Miller, Karl Krauth, Benjamin Recht, and Ludwig Schmidt. 2020. The effect of natural distribution shift on question answering models. In *International Conference on Machine Learning*. PMLR, 6905–6916.

[41] John Miller, Karl Krauth, Benjamin Recht, and Ludwig Schmidt. 2020. The Effect of Natural Distribution Shift on Question Answering Models. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 641, 12 pages.

[42] Amit Mishra and Sanjay Kumar Jain. 2016. A survey on question answering systems with classification. *Journal of King Saud University-Computer and Information Sciences* 28, 3 (2016), 345–361.

[43] Emmanuel Mutabazi, Jianjun Ni, Guangyi Tang, and Weidong Cao. 2021. A review on medical textual question answering systems based on deep learning approaches. *Applied Sciences* 11, 12 (2021), 5456.

[44] J. R. Norris. 1997. *Markov Chains.* Cambridge University Press. https://doi.org/10.1017/CBO9780511810633

[45] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*. PMLR, 4901–4911.

[46] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html

[47] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. *arXiv preprint arXiv:1806.03822* (2018).

[48] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[49] Amrita Saha, Vardaan Pahuja, Mitesh M. Khapra, Karthik Sankaranarayanan, and Sarath Chandar. 2018. Complex Sequential Question Answering: Towards Learning to Converse over Linked Question Answer Pairs with a Knowledge Graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence* (New Orleans, Louisiana, USA) *(AAAI'18/IAAI'18/EAAI'18)*. AAAI Press, Article 87, 9 pages.

[50] Xiangzhong Shen, Jieyi Zhang, Xiaonan Wang, Hongfang Yu, and Gang Sun. 2021. Deep Learning Framework Fuzzing Based on Model Mutation. In *2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC)*. IEEE, 375–380.

[51] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 303–314. https://doi.org/10.1145/3180155.3180220

[52] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2016. Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830* (2016).

[53] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. *DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks.* Association for Computing Machinery, New York, NY, USA, 146–157. https://doi.org/10.1145/3293882.3330579

[54] Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 conference on empirical methods in natural language processing*. 2013–2018.

[55] Munazza Zaib, Wei Emma Zhang, Quan Z Sheng, Adnan Mahmood, and Yang Zhang. 2021. Conversational question answering: A survey. *arXiv preprint arXiv:2106.00874* (2021).

[56] Xufan Zhang, Jiawei Liu, Ning Sun, Chunrong Fang, Jia Liu, Jiang Wang, Dong Chai, and Zhenyu Chen. 2021. Duo: Differential Fuzzing for Deep Learning Operators. *IEEE Transactions on Reliability* 70, 4 (2021), 1671–1685.

[57] Xin Zhang, An Yang, Sujian Li, and Yizhong Wang. 2019. Machine reading comprehension: a literature review. *arXiv preprint arXiv:1907.01686* (2019).

[58] Wujie Zheng, Wenyu Wang, Dian Liu, Changrong Zhang, Qinsong Zeng, Yuetang Deng, Wei Yang, Pinjia He, and Tao Xie. 2019. Testing Untestable Neural Machine Translation: An Industrial Case. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 314–315. https://doi.org/10.1109/ICSE-Companion.2019.00131