Improving QA System Testing Efficiency Through White-Box Test Prioritization

Hanying Shao[1], Zishuo Ding[2], Kundi Yao[1,*], Haonan Zhang[1], and Weiyi Shang[1]
[1]Univeristy of Waterloo, Waterloo, ON, Canada
[2]The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, Guangdong, China
{h9shao, kundi.yao, haonan.zhang, wshang}@uwaterloo.ca, zishuoding@hkust-gz.edu.cn
*corresponding author

Abstract—Effective testing of sequence-to-sequence (seq2seq) models, such as those used in question answering (QA) systems, is essential for ensuring their reliability. While recent efforts have introduced metamorphic testing strategies to detect bugs without requiring ground-truth labels, the efficiency of these methods remains limited by their lack of test case prioritization. Executing all test cases uniformly can lead to wasted resources and slower fault discovery. In this paper, we propose a white-box prioritization framework that ranks test cases based on internal signals extracted from the underlying model. Building upon a prior work that introduced two white-box techniques (i.e., GRI and WALI) for identifying vulnerable tokens, we adapt these techniques to the task of test prioritization. Instead of generating new test inputs, our methods analyze test cases produced by QAQA and prioritize those most likely to uncover faults. We evaluate our approaches on three widely-used QA datasets: BoolQ, NarrativeQA, and SQuAD2. Experimental results show that GRI significantly improves the rate of bug detection under constrained testing budgets, while WALI achieves comparable performance to baseline methods. Our findings demonstrate the value of incorporating white-box insights into the prioritization process, offering a more efficient and effective way to test QA systems.

Keywords–Natural Language Processing; Software testing; Test prioritization; Question Answering systems

1. Introduction

The increasing integration of natural language processing (NLP) systems into critical applications, such as digital assistants and question answering (QA) platforms, demands rigorous software quality assurance. Sequence-to-sequence (seq2seq) models, widely adopted for tasks like machine translation and QA, introduce a unique set of challenges for testing due to their complex, non-deterministic behavior and open-ended outputs. To address this, metamorphic testing strategies such as QAQA [26] have emerged, which validate system robustness by introducing controlled perturbations to the input and analyzing the resulting changes in output. However, a common limitation among these approaches is the lack of test prioritization. In real-world scenarios where test execution is costly or time-sensitive, running test cases in an arbitrary or fixed order often leads to inefficient use of testing resources.

In this work, we focus on the problem of *test case prioritization* for QA systems: how to rank and execute test cases in an order that maximizes the rate of bug detection. While traditional test prioritization strategies in software engineering have shown promise in reducing cost and effort, such ideas remain underexplored in the context of NLP system testing. Existing approaches like QAQA treat all test inputs equally, neglecting the fact that certain test cases are inherently more likely to expose model faults than others. By leveraging the internal signals of the underlying model, we aim to guide the execution order of test cases toward those most likely to reveal bugs early.

This paper builds upon our prior work [25], which introduced two white-box techniques for test input generation: **GRI** (GRadient Information) and **WALI** (Word ALignment Information). In that work, GRI used gradient information to identify tokens in the input that contribute most to the model's prediction, while WALI utilized attention-based alignment to locate low-confidence alignments between input and output tokens. These techniques proved effective for generating test cases that exposed translation and QA system failures.

In this follow-up study, we extend the use of GRI and WALI from test generation to *test prioritization* in QA system testing. Rather than generating new test cases, we take the test suite produced by QAQA as input and apply white-box analysis to prioritize the execution of these cases. For GRI, we score test cases based on the gradient-based importance of inserted sentence content relative to the original question. For WALI, we analyze alignment matrices between the inserted sentence and the predicted answer to quantify relevance. Our hypothesis is that test cases with higher semantic or alignment-based relevance to sensitive tokens are more likely to trigger errors. To evaluate our proposed prioritization strategies, we apply them to QA test suites generated from three benchmark datasets: BoolQ [5], NarrativeQA [17], and SQuAD2 [23], using a Transformer-based QA model [1]. We compare our methods against the original QAQA approach using the area under the curve (AUC) of fault detection over test execution budget as our primary metric. Our results show that GRI significantly improves the efficiency and effectiveness of bug discovery compared to the baseline, while WALI performs comparably. These findings support the utility of white-box information in guiding test prioritization for NLP systems.

The contributions of this paper are as follows:

- We propose a novel application of two white-box strategies, GRI and WALI, for prioritizing test cases in QA software testing.
- We show that gradient-based prioritization (GRI) can substantially increase bug detection efficiency compared

2693-9177/25/$31.00 ©2025 IEEE
DOI 10.1109/QRS65678.2025.00046

to existing black-box strategies.
- We provide an empirical evaluation of prioritization methods on standard QA benchmarks, revealing key insights into the value and limitations of internal model signals for guiding test execution.

Our work demonstrates the effectiveness of incorporating white-box techniques into the test prioritization process, contributing to more efficient and targeted NLP system testing. All code, data, and replication materials are publicly available[1].

Paper organization. The rest of the paper is constructed as follows: Section 2 presents some background information about two white-box techniques used in our approach. Section 3 illustrates our approach of applying the white-box approaches to test case prioritization for QA software. Section 4 describes our experimental methodology. Section 5 presents results and analyzes the effectiveness of our proposed approach. Section 6 discusses threats to validity. Section 7 presents the prior studies related to this work. Finally, Section 8 concludes with a summary of contributions and future research directions.

2. Preliminary: white-box approaches

The test case prioritization approach presented in this paper leverages two white-box analysis techniques, GRI and WALI, originally introduced in our prior work [25] primarily for test generation on testing machine translation. To provide the necessary foundation for understanding their application to QA test prioritization in this work, we briefly summarize the concepts of these methods below.

2.1 Gradient-based Information (GRI)

Gradient information, indicating the sensitivity of model output to input changes, is used in deep learning for tasks like adversarial example generation [8, 10, 11, 22, 34] and identifying critical input features [12, 18]. While common for classification systems, its application to QA testing is limited. Treating QA as a sequence of multi-class classification problems (predicting tokens based on source and previous tokens), we hypothesize that gradients can identify vulnerable tokens in QA. GRI uses gradient information for this purpose. Inspired by Li et al. [18], we assume tokens with higher gradients are more likely to alter system output upon replacement.

GRI calculates the partial derivative of the loss L with respect to each input token x_i to obtain gradients. This is given by:

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{|\mathcal{V}|} \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{1}$$

Where y_j is the output for the j-th element in the output vocabulary $\mathcal{V}$, and $|\mathcal{V}|$ is the size of the output vocabulary.

Using automatic differentiation, we compute the Jacobian matrix for the loss concerning the input sentence. The resulting gradient vector highlights input tokens whose perturbation is expected to significantly alter the model's output, guiding prioritization.

[1]https://github.com/senseuwaterloo/NMT-Testing

2.2 Word Alignment-based Information (WALI)

Word alignment identifies correspondences between source and target words in a bitext, benefiting tasks like neural machine translation and grammatical error correction [20, 27, 32, 35]. WALI adopts this to identify vulnerable source tokens. Unlike GRI, WALI first identifies vulnerable target tokens and then uses alignment information to map them back to source tokens. If a target token y_i is vulnerable and aligned to the source token x_j, then x_j is deemed vulnerable.

We use a Transformer model, which relies on attention, for testing [29, 30], following prior work [2, 9, 32]. Attention weights $\boldsymbol{\alpha}_{i,j}$ from the Transformer provide alignment scores between target token y_i and source token x_j:

$$\boldsymbol{\alpha}_{i,j} = \textit{softmax}\left(\frac{Q_i K_j^T}{\sqrt{d_k}}\right) \tag{2}$$

where Q_i is the query matrix, K_j is the key matrix, $\sqrt{d_k}$ is a normalization factor where d_k is the dimension of the key/query matrix. Based on the extracted the attention weights $\boldsymbol{\alpha}$, the alignment matrix $\boldsymbol{A}$ is then calculated as:

$$\boldsymbol{A}_{i,j}(\boldsymbol{\alpha}) = \begin{cases} 1 & j = \arg\max_{j'} \boldsymbol{\alpha}_{i,j'} \\ 0 & otherwise \end{cases} \tag{3}$$

where $\boldsymbol{A}_{i,j} = 1$ indicates y_i is aligned to x_j.

For both machine translation and QA evaluation, we employ a transformer-based sequence-to-sequence model. This allows leveraging encoder-decoder attention layers to map input and output tokens, identifying candidates for replacement in test generation and bug-revealing test cases for prioritization.

3. Approach

This section details how two white-box methods prioritize generated QA test sentences. Our approach focuses on sentences strongly correlated with vulnerable tokens, which are hypothesized to be more bug-revealing, to streamline testing by emphasizing impactful test cases.

3.1 Overview

Figure 1 outlines our proposed approaches for testing QA software. Initially, for each source sentence, we identify related sentences from the training dataset using methods from baseline research [26]. Next, we create a new input by adding a redundant sentence to either the question or the context, following a consistent strategy. Within our white-box framework, we employ two specific strategies to order the test cases: (1) identifying vulnerable tokens based on gradient information (GRI) and (2) identifying them through word alignment (WALI). Subsequently, we organize the new test inputs by ranking the test cases according to the relevance of the selected sentences and the targeted tokens, specifically ordering them by the maximum gradient or attention values associated with the selected tokens within the sentences. This reordering of test cases ensures they are executed in a prioritized manner. The QA software then processes these reordered sentences (test

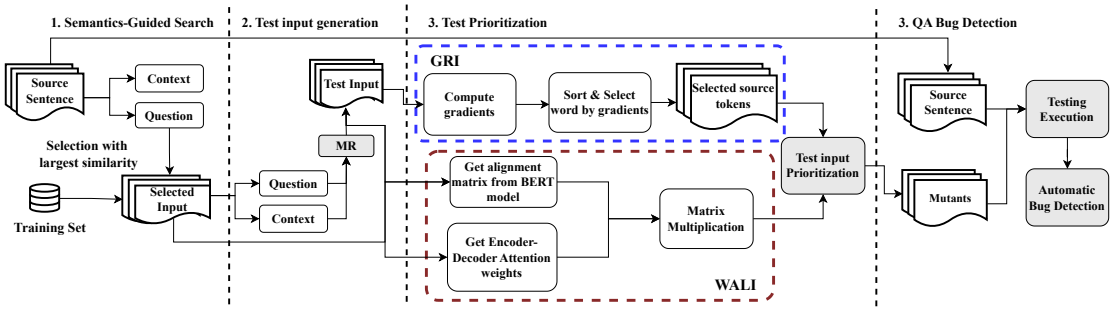

Figure 1: An overview of our approach.

cases or mutants), generating corresponding responses. If the variance between a new response and the original response surpasses a predetermined threshold, it indicates a potential bug.

3.2 GRI for test prioritization

We reorder QA software test cases using question token gradients presented in Section 2-A to enhance bug detection. In particular, we identify top-k tokens in the original question by gradient magnitude using the baseline's semantics-guided search. Test cases are then prioritized by the maximum gradient of these top-k tokens found within the inserted sentence (or 0 if these tokens are absent). This approach leverages the inserted sentence's gradient-derived relevance, based on the hypothesis that higher relevance is more likely to confuse the QA software, even if question semantics are preserved. Algorithm 1 details this GRI-based prioritization process.

Algorithm 1: GRI Prioritization

Input: a source sentence $\mathbf{x}$, and the QA system $\mathcal{F}(\cdot)$ and tokenizer *Tokenizer*
Output: a set of reordered sentences $\mathbf{X}'$

begin
  $tokens \leftarrow$ `Tokenizer(`$\mathbf{x}$`)`
  $embeddings \leftarrow \mathcal{F}(tokens)$
  $output \leftarrow \mathcal{F}(embeddings)$
  $G \leftarrow$ `GetGradient(`$output$`)`
  $G_{sorted} \leftarrow$ `Sort(`G`)`
  $T_{ordered} \leftarrow$ `SortTokensbyGrad(`$tokens$`)`
  $S_{selected} \leftarrow$ `Semantics-GuidedSearch(`*Training Set*`)`
  $g_{max} \leftarrow 0$
  foreach $w_i, g_i \in T_{ordered}$ **do**
    if $w_i \in S_{selected}$ **then**
      $g_{max} \leftarrow \max\{g_{max}, g_i\}$
  $\mathbf{X}'_{ordered} \leftarrow$ `SortbyMaxGradient(`$\mathbf{X}'$`)`
  return $\mathbf{X}'_{ordered}$

3.3 WALI for test prioritization

As explained in section 2-B, attention weights in the transformer model illustrate the connections between target and source tokens. Specifically, question-answering scenarios usually present an intricate token mapping due to the disproportionate lengths of source inputs and outputs. Consequently, the emphasis shifts toward analyzing the alignment weights that link target and source tokens.

To enhance the identification of bug-revealing test sentences, the hypothesis is that a stronger correlation between the inserted sentence and the answer increases the likelihood of influencing the answer. Thus, the goal is to assess the correlation between the inserted sentence and the answer. While the transformer model provides attention weights showing input-output token mappings, it does not directly offer the correlation with inserted sentences. The proposed method involves using these attention weights, as detailed in section 2-B, to derive the attention matrix for the input sentences (question and context) against the output answer. An alignment matrix is then calculated for the original input and the inserted sentence using a pre-trained BERT model [6]. The BERT model is also a transformer-based model pre-trained on a large corpus of multilingual data that can provide the alignment between the input and output sentences. By matrix multiplication of these two matrices, a new matrix emerges, illustrating the relationship between the selected tokens and the answer, aiding in prioritizing the test cases effectively.

To mathematically formalize the algorithm described for prioritizing bug-revealing test sentences, consider the following equations:

Let $A \in \mathbb{R}^{m \times n}$ represent the attention matrix from the transformer model, where m is the number of input tokens and n is the number of output tokens (answer). An element a_{ij} in A indicates the attention weight from the i-th input token to the j-th output token.

$$A = [a_{ij}] \quad \text{for } i = 1, \ldots, m \text{ and } j = 1, \ldots, n$$

Let $B \in \mathbb{R}^{m \times o}$ denote the alignment matrix obtained using the BERT model [6], where o is the number of tokens in the inserted sentence. An element b_{ik} in B represents the alignment score between the i-th input token and the k-th token in the inserted sentence.

$$B = [b_{ik}] \quad \text{for } i = 1, \ldots, m \text{ and } k = 1, \ldots, o$$

The correlation matrix $C \in \mathbb{R}^{o \times n}$ is computed by multiplying matrices A and B, where an element c_{kj} in C corresponds to the correlation between the k-th token in the inserted sentence

and the j-th token in the output answer.

$$C = B^{\top} \times A$$

$$C = [c_{kj}] \quad \text{for } k = 1, \ldots, o \text{ and } j = 1, \ldots, n$$

This product $C = B^{\top} \times A$ provides a detailed view of how tokens in the inserted sentence is related to tokens in the answer. Figure 2 illustrates the matrix multiplication between A and $B^{\top}$.

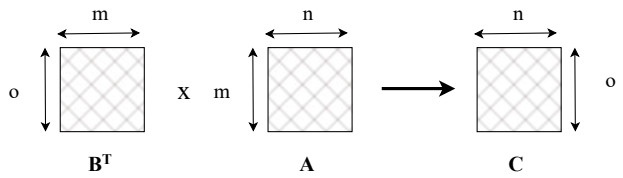

Figure 2: Matrix multiplication where A represents the attention matrix, m is the number of input tokens and n is the number of output tokens, B denote the alignment matrix obtained using the BERT model [6], o is the number of tokens in the inserted sentence.

The magnitude of the correlation matrix is utilized to sequence the test sentences, prioritizing those that have a higher likelihood of revealing bugs during testing. Test sentences are ordered in descending order according to the magnitude of the correlation matrix, ensuring that those with the greatest potential impact are tested first.

3.4 Test input generation

For the test input generation, we follow the methodology from a previous study. The process, known as QAQA, involves searching the training data for a sentence that closely resembles the original question in terms of semantic content. To identify the core semantics of the question, it is first condensed using the compression model SLAHAN [13]. Subsequently, both the truncated question and the training data sentences are transformed into embeddings using Sentence-BERT (SBERT) [24]. By computing the cosine similarity between these embeddings, the sentence from the training set with the highest similarity score to the truncated question is chosen for mutation. This selected sentence is then used to generate a new test input.

To generate mutants, QAQA leverages five metamorphic relations: Equivalent Question (EQ), Equivalent Context (EC), Test Integration (TI), Equivalent Question & Context (EQC), and Equivalent Test Integration (ETI). These relations guide how the selected sentence is inserted into the question and/or context to create a new test input. We employ this same methodology to produce mutants in our testing strategy. For a more detailed explanation of these metamorphic relations and their application in the mutant generation, refer to the study detailed in [26].

3.5 Bug detection

The bug detection approach aligns with the baseline method to ensure comparability, focusing on the consistency of metamorphic relations. Answers provided by the QA software are compared; if the discrepancy between the answer to the generated input and the original exceeds a predefined threshold, it signals a potential bug. Considering the linguistic variability where identical semantics can be presented differently (e.g., "UK" vs. "United Kingdom"), the approach also includes measuring the embeddings' similarity to ensure semantic consistency between answers. Phrase-Bert (PBERT) [33], an advanced model for phrase-level representation, is employed to convert the answers into embedding vectors, followed by computing cosine similarity to detect semantic variations. When the similarity score falls below the established threshold (0.76, consistent with the baseline methodology [26]), a bug is reported. More details can be found in prior research on QA software testing [26].

3.6 Test prioritization evaluation metric

To assess the effectiveness of test prioritization, it is essential to have an evaluation metric. A common metric is a graph depicting the percentage of detected faults versus the fraction of the test suite used. This curve illustrates the cumulative percentage of faults detected as testing progresses. The shaded area under this curve signifies the weighted average of the percentage of faults detected throughout the lifespan of the test suite. This area is referred to as the prioritized test suite's average percentage of faults detected.

In our experiments, we will evaluate the area under the curve (AUC) of the percentage of detected faults during test execution. We will compare this with the ideal scenario, where all bugs are scheduled in priority, to assess the performance of our white-box approaches (Figure 3 provides an example of the curves). This comparison will allow us to quantify how effectively our approaches can identify and prioritize critical test cases that are likely to reveal faults, thereby optimizing the testing process and ensuring more efficient resource utilization.

4. Experimental Setup

In this section, we present our experimental setup, including the QA dataset and software for evaluation, and the experimental environments.

4.1 Dataset

The study employs three well-regarded QA datasets previously utilized in existing research. The baseline QAQA used these datasets as well for evaluation; therefore, we adopted the same datasets to maintain consistency in comparison. The training datasets are used to build the UnifiedQA model, while the test sets are used as test cases for the proposed approaches. The details of the datasets are listed below.

BoolQ. BoolQ [5] is a Boolean QA dataset, where the answers to the questions are either "yes" or "no". This dataset comprises questions sourced from Google search engine queries, with corresponding contexts extracted from Wikipedia articles.

SQuAD2. SQuAD2 [23] is an extractive QA dataset in which the answer is a segment of text. The questions and answers are collected from Wikipedia articles. SQuAD2 includes unanswerable questions, meaning that some questions are designed with "<No Answer>" indicating that the text does not contain information to answer the query.

NarrativeQA. NarrativeQA [17] is an abstractive QA dataset in which the answer is not a span of the context. The dataset is collected from books and movie scripts to test the comprehension of the model in the context.

4.2 QA software

We used the pre-trained T5-large-based UnifiedQA [1] model as the testing subject, which aligns with the baseline approach, QAQA. It is one of the state-of-the-art models that is proven to perform well across diverse datasets. In addition, it was widely studied in prior researches [4, 26] as well as an experimental subject; employing this model facilitates direct comparisons with prior results.

4.3 Implementation settings

In this experiment, we improve the testing efficiency using GRI and WALI for prioritization. For the purpose of a fair comparison, we adopt the same approach as in the previous study [26] for generating new test sentences using metamorphic relations for each original source sentence. We conducted the experiment on Ubuntu 18.04 with an NVIDIA GTX 1080Ti GPU.

5. Evaluation

In this section, we evaluate our proposed approaches against the SOTA testing baseline (i.e., QAQA). The effectiveness of our approach can be evaluated by the ability to identify the bug-revealing test sentences and position these potential bug-triggering sentences at the beginning of the testing sequence, thus improving the fault detection rates over the testing phase. Specifically, we aim to answer the following research questions (RQs):

RQ1: How effective are the white-box approaches in prioritizing the bug-revealing test cases?

Motivation. Extensive approaches have been proposed for testing question answering systems, while few consider conducting the testing from a white-box perspective. Meanwhile, studies [11, 18, 32] have shown that using white-box methods can benefit many NLP tasks. Therefore, in this RQ, we would like to explore whether our two white-box-based approaches (i.e., GRI and WALI) can improve the AUC (Area Under Curve) of the cumulative count of bugs reported during the testing, which indicates the rate of fault detection of the prioritization techniques.

Approach. To answer this research question, we apply GRI and WALI as well as the baseline approaches on each source sentence in the BoolQ, NarrativeQ, and SQuAD2 datasets. With the generated mutants, we then examine whether the mutants can reveal a bug (cf. Section 3-E). For quantitative evaluation, we focus on the number of bugs detected using the similarity metrics over the number of test cases executed. To ensure the validity of the results, following previous work [26], we randomly sample 100 test cases for manual evaluation.

TABLE I: AUC for different approaches

	BoolQ	**NarrativeQA**	**SQuAD2**
QAQA	51.37	51.04	50.19
GRI	**58.24**	**55.54**	**52.68**
Ideal	89.72	84.59	80.86

Result. **Our proposed approaches, GRI, can outperform the baseline approaches under identical experimental settings.** The results comparing GRI and WALI with baseline methods are shown in Figure 3. This figure demonstrates the cumulative bugs detected by different strategies against the number of test cases executed in various sequences across three datasets. It's evident that GRI surpasses the baseline in two key areas: (1) **it uncovers more bugs within a specific testing budget**, and (2) **it achieves set testing goals with fewer resources post-prioritization**. For instance, GRI detects 315 bugs in BoolQ with 1,000 test cases, surpassing the 221 bugs identified by QAQA, as illustrated in Figure 3(a). Moreover, to find 200 bugs, GRI requires 522 tests, whereas QAQA needs 890 tests. This figure illustrates the effectiveness of GRI in identifying test cases with a higher likelihood of revealing bugs, thereby enhancing the testing process's efficiency through prioritization.

Additionally, GRI consistently shows a higher percentage of AUC when set against the QAQA baseline. The AUC (Area Under the Curve) in Figure 3 illustrates the rate at which bugs are discovered as testing progresses. A higher AUC indicates that a larger number of bugs are identified early in the testing cycle, suggesting that the testing process is effective in quickly uncovering faults in the software. Table I displays these comparisons, with the best results in bold. Specifically, GRI outperforms the baseline approach in identifying more bugs within the same testing budget across BoolQ, NarrativeQA, and SQuAD2 datasets, showing an increase in AUC percentage of 6.87%, 4.50%, and 2.49%, respectively. This emphasizes GRI's effectiveness in exploiting QA systems' vulnerabilities by using gradient information, thus enhancing testing success rates.

From Figure 4, it is evident that the performance of WALI is not particularly impressive. The curve and the Area Under the Curve (AUC) of WALI are quite similar to those of the original QAQA curve. For the SQuAD2 dataset, the percentage of AUC is only about 5% higher than the original baseline approach. However, since this improvement is not consistent across all three datasets and is relatively modest, it is not sufficient to conclude that WALI significantly enhances detection efficiency. Thus, the prioritization based on the WALI approach does not offer a significant improvement over the established QAQA method. However, our proposed approach offers the potential to explore the relationship between the input and output using the alignment information of the sequence-to-

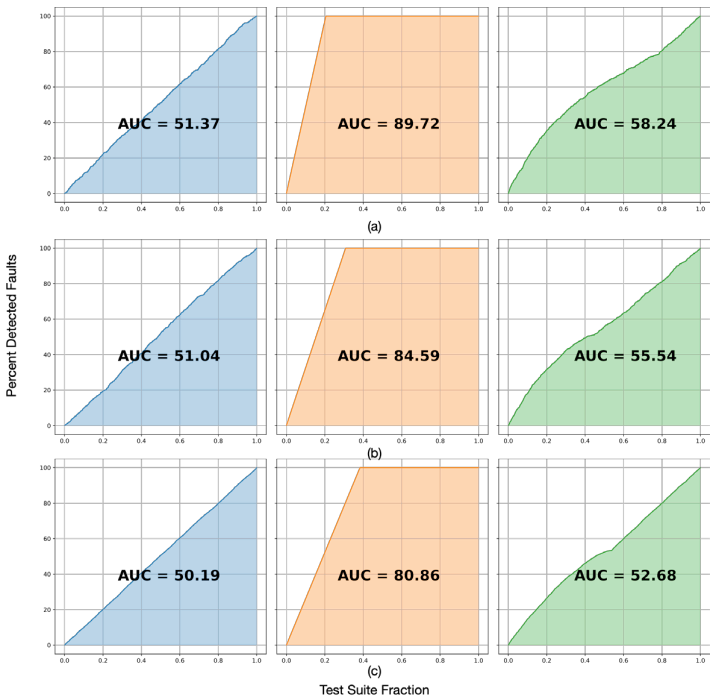

Figure 3: Percentage of detected bugs(y-axis) vs. fraction of test suite executed(x-axis) using the initial test order, the ideal scenario, and GRI-based prioritization for (a) BoolQ, (b) NarrativeQA and (c) SQuAD2 dataset respectively.

TABLE II: Manual evaluation results.

Dataset	NarrativeQA	SQuAD2
Precision	79.49%	86.67%
Recall	96.87%	83.87%
F1 Score	87.32%	85.24%

sequence models.

As the test oracle depends on the similarity score between the original and new answers, ensuring the validity of the results necessitates a manual evaluation. Given that the original test generation methodology of QAQA remains unchanged, the precision of the test oracle should remain consistent. To validate the results, we employed the same evaluation approach, sampling 100 test cases from the NarrativeQA and SQuAD2 datasets. These test cases and their results were manually labeled to assess accuracy. The precision, recall, and F1-score of this evaluation are presented in Table II.

From Table II, the precision, recall, and F1 score align closely with those reported in the earlier study [26], demonstrating the test oracle's reliability. Specifically, in the NarrativeQA dataset, there were 8 false positives and 1 false negative out of 100 random samples. For the SQuAD2 dataset, the evaluation revealed 4 false positives and 5 false negatives among the 100 samples. Additionally, the QAQA report noted 5 false positives and 27 false negatives from 300 samples of generated question and answer pairs. By comparing these outcomes with the false positives and false negatives documented in the paper [26], it is clear that our testing strategy maintains its accuracy. This consistent performance results from our adherence to the original test generation process and the use of the same threshold values, ensuring robust and reliable testing across different datasets. Thus, the validity of the results is maintained.

Prioritizing tests with GRI quantitatively surpasses the QAQA baseline while maintaining human-validated accuracy; WALI performs comparably to the baseline. This demonstrates the potential for future research into white-box methods to enhance QA test prioritization and efficiency.

RQ2: What are the characteristics of the test cases that are not identified by GRI and WALI?

Motivation. While GRI and WALI have effectively identified several test cases more likely to uncover bugs, there are still instances where test cases evade detection. This research question (RQ) aims to investigate the characteristics of the test cases that GRI and WALI failed to identify and to explore the factors that may increase the likelihood of generating bug-revealing test cases for QA systems.

Approach. To address this research question, we analyze test cases that reported bugs despite low gradient or attention mapping values. Our investigation includes an in-depth analysis of source input length to determine its influence on bug detection effectiveness and its correlation with the probability of a test case revealing a bug. This aims to refine and enhance QA

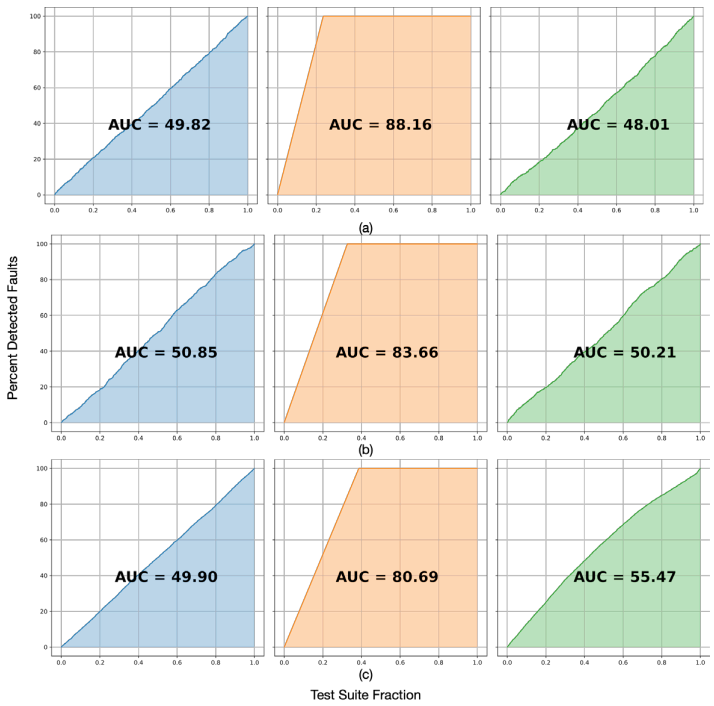

Figure 4: Percentage of detected bugs(y-axis) vs. fraction of test suite executed(x-axis) using the initial test order, the ideal scenario, and WALI-based prioritization for (a) BoolQ, (b) NarrativeQA and (c) SQuAD2 dataset respectively.

system testing strategies.

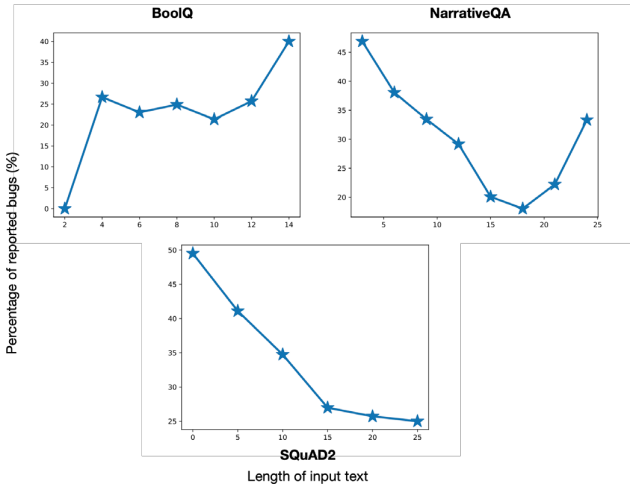

Figure 5: Percentage of reported bugs (y-axis) vs. length of the input sentence (x-axis). This figure shows the distribution of bugs in terms of sentence length.

First, we examine the correlation between the length of the question and the likelihood of reporting a bug after a mutation is inserted. Figure 5 illustrates a negative relationship between the percentage of bugs reported and the length of the question in the input across all datasets except BoolQ. This trend arises because shorter questions are more significantly affected by the addition of redundant sentences to the input question and context, logically leading to a negative relationship between the likelihood of bug reporting and question length. For the dataset NarrativeQA, the Spearman correlation coefficient [19] ρ is -0.69 with a p-value of 0.057, which indicates a strong negative correlation between the variables. This finding implies that shorter questions in NarrativeQA are particularly susceptible to generating detectable bugs when altered with test case mutations.

The exception observed with the BoolQ dataset can be attributed to the binary nature of its answers and the consistent structure of boolean questions. Typically, boolean questions demand a straightforward "yes" or "no" response, which may not vary significantly with changes in question length. This consistency in the question format and answer type reduces the variability in how questions are processed and understood, potentially diminishing the impact of additional or redundant information introduced through mutations. Consequently, the correlation between question length and bug occurrence that is evident in other datasets may not manifest as strongly in BoolQ.

In Table 6, two test cases are presented where GRI and WALI did not effectively prioritize as bug-revealing, yet they reported bugs. Observing these examples, it is evident that the added sentences do not initially correlate with the question. However, when incorporated into the question using the QAQA methodology—prefaced with phrases like “I have known” or “I have heard”—they are likely to create an implication that

Original Question	Added Sentence	Original Answer	New Answer
What is another name for the goddess diana?	I have known that pauline admires juliet's outspoken arrogance and beauty	Cynthia	juliet is diana
What is the name of the leader of the gang that is laying siege to the oil refinery?	I heard a whisper that Gabriel Syme gives a rousing anarchist speech and wins the vote	Lord Humungus	pappagallo

Figure 6: Example of test cases that reported a bug but were not identified by GRI and WALI.

influences the answer.

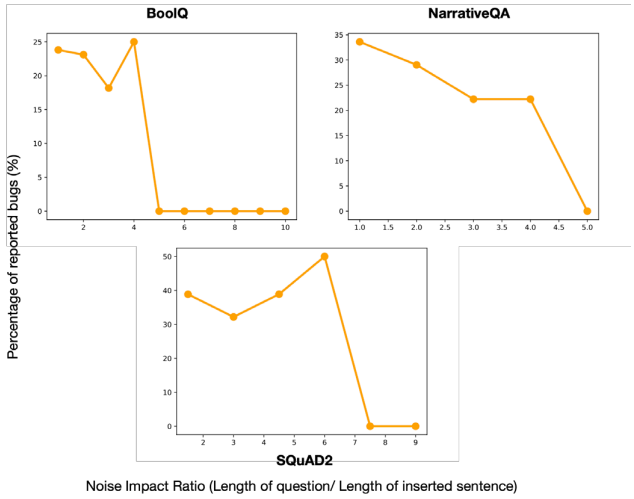

Figure 7: Percentage of reported bugs (y-axis) vs. length of question/ length of inserted sentence (x-axis). This figure shows the distribution of bugs in terms of the ratio between texts.

For the first example, GRI identified "*diana*" and "*goddess*" as significant tokens within the question. Upon analyzing the keywords in the added sentence, it failed to find any direct correlation with these identified tokens. WALI calculated a score of 0.45 for the magnitude of the correlation matrix of the added sentence and the answer, suggesting a moderate connection. Despite the initial absence of a direct link, the way the sentence was integrated and the phrase, "*juliet's outspoken arrogance and beauty*" imply a semantic connection. This integration subtly misled the model, causing it to provide an incorrect response. This outcome underscores the complexity of detecting and interpreting the semantic implications of added sentences in QA systems, highlighting areas for improvement in the tools used to prioritize potential bug-revealing test cases. In the second example, there is a potential implication established even though there is no direct correlation or similarity between the added sentence and the question. However, while the answer changes, it is not influenced by the added sentence but rather presents a completely different response. This scenario is another typical example of test cases that GRI and WALI fail to identify. The addition of a new sentence to the input text introduces noise and can impact the model's output. To explore this effect further, we analyze whether the ratio of the length of the question to the length of the added sentence correlates with the number of bugs reported. We hypothesize that the longer the added sentence relative to the shorter original question, the greater the impact of noise on the model.

From Figure 7, we observe a negative correlation between the ratio, which we will refer to as the "Noise Impact Ratio" (Length of question/ Length of inserted sentence), and the number of reported bugs. This metric helps us quantify how the proportion of added content to original content influences the likelihood of generating bugs, underscoring the sensitivity of the QA model to variations in input length. This observation emphasizes the importance of the length of the inserted sentence in relation to the original input text. A lengthy inserted sentence can introduce significant noise, causing the model to produce a different answer even if there is no semantic correlation between the texts. In such cases, the model struggles to accurately comprehend the question and identify the crucial segments, leading to incorrect responses.

> Reported bugs negatively correlate with source sentence length and also with the ratio of question length to inserted sentence length. Although the question and added sentence may not be directly similar, their combination can introduce misleading implications for the model, leading to incorrect answers.

6. Threats to Validity

In this section, we discuss the threats to the validity of our research.

External Validity. In our study, we have utilized three datasets (i.e., BoolQ [5], NarrativeQA [17] and SQuAD2 [23]) and a QA software (i.e., UnifiedQA [1]). We followed strictly the experimental setup of QAQA in the research to ensure a fair comparison. Therefore, it is necessary for future research to examine the effectiveness of our proposed approaches on other QA models and datasets.

Internal Validity. In WALI, we use the encoder-decoder attention weights of the UnifiedQA model to align the target and source tokens, which although effective, may not always be precise. For future research, we can employ a more sophisticated alignment approach to enhance the approach. In addition, the automatic test oracle used for bug detection relies on cosine similarity of embedding vectors to evaluate the quality of answers. However, the metrics used for evaluating output answers may not accurately reflect human perceptions of quality, and human evaluation may be subject to individual biases. We followed the same human evaluation criteria as used in the baseline approaches and randomized the test inputs to reduce bias. To address this issue, future research could incorporate more manual evaluations to assess the effectiveness of GRI and WALI with a larger number of evaluators to minimize bias.

7. Related work

In this section, we present the related work in test selection and prioritization, and QA system testing.

7.1 Test selection and prioritization techniques

Test case prioritization aims to improve testing efficiency by running fault-detecting test cases earlier [7]. Machine learning approaches are commonly leveraged to address this problem [21]. Examples include reinforcement learning using test failure as a reward [28], unsupervised learning to group test cases by coverage [14], and supervised learning models like Random Forest and Rank Boost [3]. NLP topic modeling has also been used for test case vector representation and prioritization via a greedy algorithm [31]. However, these techniques often depend on source code coverage [14], historical failures [28], or structured test descriptions [31]. Their direct application to complex AI systems like QA is challenging due to unstructured natural language inputs, semantic correctness criteria, and testing methodologies [26] that don't easily produce traditional coverage or failure logs. This gap highlights the need for prioritization techniques tailored to QA systems.

7.2 Question answering system testing

QA systems, significantly advanced by machine learning and deep learning, aim to answer human questions in natural language. Reference-based evaluation, using benchmarks like SQuAD2 [23], BoolQ [5], BoolQ-NP [16], NarrativeQA [17], and MultiRC [15], has been a primary testing methodology. However, this approach is labor-intensive due to manual labeling and ineffective for the vast number of unlabeled, real-world user questions [4, 26]. To address these limitations, automated, label-free methods like QAAskeR [4] and QAQA [26] have emerged. Both employ metamorphic testing. QAAskeR generates new question-answer pairs from initial outputs, flagging inconsistencies as potential bugs. QAQA applies sentence-level mutations (e.g., adding redundant sentences) to original questions or contexts, expecting consistent answers. While these methods overcome the manual labeling dependency, they are black-box approaches that execute test cases without prioritizing those more likely to find bugs. Our white-box approaches, GRI and WALI, address this by prioritizing test cases for sequence-to-sequence transformer-based QA models (like that in QAQA). By identifying unstable tokens in source sentences, these methods aim to reveal model vulnerabilities and optimize testing by prioritizing effective test cases.

8. Conclusion

In this study, we apply two white-box approaches, GRI and WALI, to prioritize test cases generated by QAQA for QA software testing, aiming to enhance test execution efficiency. Results show GRI efficiently and effectively prioritizes bug-revealing test cases, outperforming the baseline, while WALI's performance is comparable. These white-box methods identify and prioritize test cases more likely to reveal bugs for early execution, thereby improving efficiency and reducing computational costs. Our research highlights advancements in using white-box approaches for testing sequence-to-sequence models and demonstrates their potential for AI software quality assurance.

References

[1] allenai, 2020. URL https://huggingface.co/allenai/unifiedqa-t5-large.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations*, 2015.

[3] Antonia Bertolino, Antonio Guerriero, Breno Miranda, Roberto Pietrantuono, and Stefano Russo. Learning-to-rank vs ranking-to-learn: strategies for regression testing in continuous integration. In Gregg Rothermel and Doo-Hwan Bae, editors, *ICSE '20: 42nd International Conference on Software Engineering*, 2020.

[4] Songqiang Chen, Shuo Jin, and Xiaoyuan Xie. Testing your question answering software via asking recursively. In *36th IEEE/ACM International Conference on Automated Software Engineering*, 2021.

[5] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, 2018.

[7] Sebastian G. Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. *IEEE Trans. Software Eng.*, 2002.

[8] Ivan Fursov, Alexey Zaytsev, Nikita Kluchnikov, Andrey Kravchenko, and Eugeny Burnaev. Differentiable language model adversarial attacks on categorical sequence classifiers. *CoRR*, 2020.

[9] Hamidreza Ghader and Christof Monz. What does attention in neural machine translation pay attention to? In Greg Kondrak and Taro Watanabe, editors, *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, 2017.

[10] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations*, 2015.

[11] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *CoRR*, 2016.

[12] Bairu Hou, Jinghan Jia, Yihua Zhang, Guanhua Zhang, Yang Zhang, Sijia Liu, and Shiyu Chang. Textgrad: Advancing robustness evaluation in NLP by gradient-driven optimization. In *The Eleventh International Conference on Learning Representations*, 2023.

[13] Hidetaka Kamigaito and Manabu Okumura. Syntactically look-ahead attention network for sentence compression. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[14] Passant Kandil, Sherin M. Moussa, and Nagwa L. Badr. Cluster-based test cases prioritization and selection technique for agile regression testing. *J. Softw. Evol. Process.*, 2017.

[15] Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018.

[16] Daniel Khashabi, Tushar Khot, and Ashish Sabharwal. More bang for your buck: Natural perturbation for robust question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020.

[17] Tomás Kociský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Trans. Assoc. Comput. Linguistics*, 2018.

[18] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. Textbugger: Generating adversarial text against real-world applications. In *26th Annual Network and Distributed System Security Symposium*, 2019.

[19] A.M. Mood, F.A. Graybill, and D.C. Boes. *Introduction to the Theory of Statistics*. McGraw-Hill, 1973.

[20] Thien Nguyen, Lam Nguyen, Phuoc Tran, and Huu Nguyen. Improving transformer-based neural machine translation with prior alignments. *Complex.*, 2021.

[21] Rongqi Pan, Mojtaba Bagherzadeh, Taher Ahmed Ghaleb, and Lionel C. Briand. Test case selection and prioritization using machine learning: a systematic literature review. *Empir. Softw. Eng.*, 2022.

[22] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: automated whitebox testing of deep learning systems. *Commun. ACM*, 2019.

[23] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[24] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[25] Hanying Shao, Zishuo Ding, Weiyi Shang, Jinqiu Yang, and Nikolaos Tsantalis. Towards effectively testing machine translation systems from white-box perspectives. *Empir. Softw. Eng.*, 2025.

[26] Qingchao Shen, Junjie Chen, Jie M. Zhang, Haoyu Wang, Shuang Liu, and Menghan Tian. Natural test generation for precise testing of question answering software. In *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022.

[27] Kai Song, Xiaoqing Zhou, Heng Yu, Zhongqiang Huang, Yue Zhang, Weihua Luo, Xiangyu Duan, and Min Zhang. Towards better word alignment in transformer. *IEEE ACM Trans. Audio Speech Lang. Process.*, 2020.

[28] Helge Spieker, Arnaud Gotlieb, Dusica Marijan, and Morten Mossige. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In Tevfik Bultan and Koushik Sen, editors, *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017.

[29] Zeyu Sun, Jie M. Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. Automatic testing and improvement of machine translation. In Gregg Rothermel and Doo-Hwan Bae, editors, *ICSE '20: 42nd International Conference on Software Engineering*, 2020.

[30] Zeyu Sun, Jie M. Zhang, Yingfei Xiong, Mark Harman, Mike Papadakis, and Lu Zhang. Improving machine translation systems via isotopic replacement. In *44th IEEE/ACM 44th International Conference on Software Engineering*, 2022.

[31] Stephen W. Thomas, Hadi Hemmati, Ahmed E. Hassan, and Dorothea Blostein. Static test case prioritization using topic models. *Empir. Softw. Eng.*, 2014.

[32] Lihao Wang and Xiaoqing Zheng. Improving grammatical error correction models with purpose-built adversarial examples. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020.

[33] Shufan Wang, Laure Thompson, and Mohit Iyyer. Phrase-bert: Improved phrase embeddings from BERT with an application to corpus exploration. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[34] Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. Generating fluent adversarial examples for natural languages. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics*, 2019.

[35] Jingyi Zhang and Josef van Genabith. A bidirectional transformer based alignment model for unsupervised word alignment. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, 2021.