COSTELLO: Contrastive Testing for Embedding-Based Large Language Model as a Service Embeddings

WEIPENG JIANG, Xi'an Jiaotong University, China
JUAN ZHAI, University of Massachusetts Amherst, USA
SHIQING MA, University of Massachusetts Amherst, USA
XIAOYU ZHANG, Xi'an Jiaotong University, China
CHAO SHEN, Xi'an Jiaotong University, China

Large language models have gained significant popularity and are often provided as a service (i.e., LLMaaS). Companies like OpenAI and Google provide online APIs of LLMs to allow downstream users to create innovative applications. Despite its popularity, LLM safety and quality assurance is a well-recognized concern in the real world, requiring extra efforts for testing these LLMs. Unfortunately, while end-to-end services like ChatGPT have garnered rising attention in terms of testing, the LLMaaS embeddings have comparatively received less scrutiny. We state the importance of testing and uncovering problematic individual embeddings without considering downstream applications. The abstraction and non-interpretability of embedded vectors, combined with the black-box inaccessibility of LLMaaS, make testing a challenging puzzle. This paper proposes Costello, a black-box approach to reveal potential defects in abstract embedding vectors from LLMaaS by *contrastive testing*. Our intuition is that high-quality LLMs can adequately capture the semantic relationships of the input texts and properly represent their relationships in the high-dimensional space. For the given interface of LLMaaS and seed inputs, Costello can automatically generate test suites and output words with potential problematic embeddings. The idea is to synthesize contrastive samples with guidance, including positive and negative samples, by mutating seed inputs. Our synthesis guide will leverage task-specific properties to control the mutation procedure and generate samples with known partial relationships in the high-dimensional space. Thus, we can compare the expected relationship (oracle) and embedding distance (output of LLMs) to locate potential buggy cases. We evaluate Costello on 42 open-source (encoder-based) language models and two real-world commercial LLMaaS. Experimental results show that Costello can effectively detect semantic violations, where more than 62% of violations on average result in erroneous behaviors (e.g., unfairness) of downstream applications.

CCS Concepts: • **Software and its engineering** → *Software testing and debugging*.

Additional Key Words and Phrases: Contrastive Testing, LLMaaS, Embeddings

ACM Reference Format:
Weipeng Jiang, Juan Zhai, Shiqing Ma, Xiaoyu Zhang, and Chao Shen. 2024. COSTELLO: Contrastive Testing for Embedding-Based Large Language Model as a Service Embeddings. *Proc. ACM Softw. Eng.* 1, FSE, Article 41 (July 2024), 23 pages. https://doi.org/10.1145/3643767

Authors' addresses: Weipeng Jiang, Xi'an Jiaotong University, Xi'an, China, lenijwp@stu.xjtu.edu.cn; Juan Zhai, University of Massachusetts Amherst, Amherst, USA, juanzhai@umass.edu; Shiqing Ma, University of Massachusetts Amherst, Amherst, USA, shiqingma@umass.edu; Xiaoyu Zhang, Xi'an Jiaotong University, Xi'an, China, zxy0927@stu.xjtu.edu.cn; Chao Shen, Xi'an Jiaotong University, Xi'an, China, chaoshen@xjtu.edu.cn.

ACM 2994-970X/2024/7-ART41
https://doi.org/10.1145/3643767

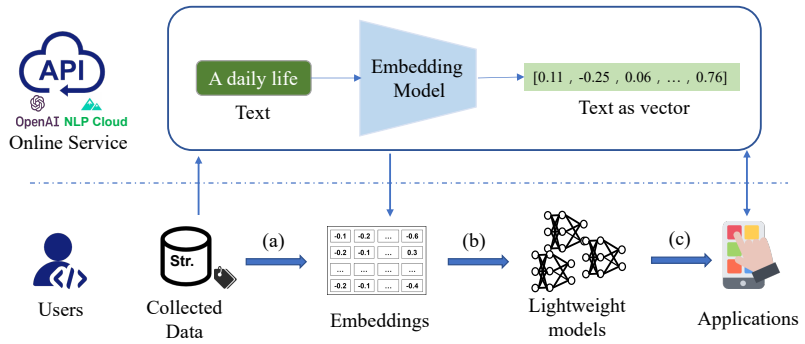

Fig. 1. An illustration of developing downstream applications with LLMaaS embeddings.

1 INTRODUCTION

Recently, large language models (LLMs) have demonstrated remarkable power and even surpassed human performance on a wide range of NLP scenarios, such as text classification [17], question answering [6], machine translation [4]. However, the cost of training a well-performing LLM often runs into hundreds of millions of dollars, leading companies to keep them proprietary for commercial reasons. Besides, running LLMs can be unaffordable for personal users due to huge GPU consumption. As a result, companies like OpenAI, Google and Baidu typically deploy their LLMs on cloud servers and release the power of LLMs as a service through black-box APIs, such as GPT-3 [9], PaLM [14] and ERNIE 3.0 [60]. Large Language Model as a Service (LLMaaS) provides users with a range of options to interact with the LLM via API and create their customized applications, including but not limited to prompt-based tuning [37, 39, 59], in-context learning [9, 54], and embedding-based approaches [48]. ChatGPT stands as one of the foremost and highly acclaimed prompt-based applications, emerging as a trailblazer within the realm of LLMaaS, boasting an extensive user base numbering in the hundreds of millions [6].

In this work, we focus on the LLMaaS providing embeddings, which acts as a feature extractor that allows users to acquire high-quality representations and subsequently do follow-up development, as illustrated in Figure 1. Firstly, users collect textual data to the online API and API feedback numeric representations (embedding vectors) in step (a). Then, in step (b), users train their own lightweight AI models based on embeddings according to their needs, such as sentiment analysis, semantic matching, etc. Finally, the lightweight models will be deployed into real applications in step (c) and provide services by interacting with the online API. Under this production paradigm, users do not need to rack their brains to construct appropriate prompts to induce LLMs to output satisfactory answers, and the LLMaaS platforms do not need to frequently fine-tune LLMs to cater various needs of users. LLMaaS embeddings are becoming increasingly popular due to good flexibility and cost-effectiveness, e.g., JetBrains achieves data source classification powered by OpenAI's embeddings [2] and FED Group makes candidate recommendations through HrFlow's embeddings [1]. Moreover, embeddings find applications for text indexing in conjunction with ChatGPT to build customized Question-Answer (QA) systems, leveraging local knowledge bases [7].

Despite its popularity, LLMaaS is far from perfect in many critical aspects (e.g., robustness, fairness, etc.), which has caused widespread concern among researchers and practitioners alike. Even ChatGPT, currently the most advanced LLMaaS with millions of users, is vulnerable to word-level adversarial attacks [65] and rife with discriminatory bias [3]. Note that although the LLMaaS embeddings are "semi-finished products", which need to be further developed to yield real value, the quality/security concerns should not be diminished. Typically, an attacker can provide a

malicious LLMaaS and leverage tainted embeddings to perform task-independent backdoor attacks on downstream applications [10]. Many studies have demonstrated that the quality of embeddings can impact the level of performance achieved by downstream applications [17, 36, 55]. As the infrastructure for supporting a wide range of downstream applications, the impact of a flawed LLMaaS would be difficult to estimate. Hence, conducting tailored testing for LLMaaS to uncover flaws/issues is of significant importance for safety and quality. Also, testing for LLMaaS embeddings is in demand from the viewpoint of software development. On one hand, pre-developing testing helps downstream developers conduct a feasibility analysis to determine if the service meets their needs and select the appropriate upstream service providers before investing resources and time in developing the application. On the other hand, if the application they develop does not perform well with certain inputs, testing on embeddings can aid in root cause analysis by diagnosing whether the provided embeddings are bad or their training procedure still needs to be improved.

However, testing LLMaaS embeddings is still not well-recognized and remains a challenging task. **Firstly**, the black box characterization dictates that we do not have access to the internal implementation as well as the weight of the LLM, which hampers our ability to verify the procedures and identify anomalous behaviors of language models through commonly white-box testing methods, such as neuron state analysis [18, 28]. **Secondly**, obtaining the *test oracle* is tough even if we have the input and output of the target system. The LLMaaS embeddings are abstract, uninterpretable, high-dimensional vectors. This makes it impossible to directly confirm the association between input and output, or to accurately predict the expected embedding values as a reference for a given input. Some existing methods are testing various NLP software, but their target output is all textual, thus enabling them to check whether the output meets the expectation or references through methods such as metamorphic testing [11, 32, 57]. Overall, how to test the LLMaaS embeddings is still an open problem.

In this paper, we propose a novel *contrastive testing* approach for testing black-box LLMaaS, namely Costello, to identify those textual inputs in which the embedding vectors are potentially problematic. The core idea of Costello is inspired by the *contrastive learning*, a widely adopted training paradigm [13, 22] that aims to make similar samples closer to each other in the feature space while the dissimilar samples are far away from each other. Similarly, our intuition is that high-quality LLMs can adequately capture the desired relationship among multiple inputs. Our idea is to synthesize contrastive samples from the aspect of semantics and compare whether embedding vectors meet expectations. We introduce the *contrastive relationships (CRs)* to declare the positive (i.e., similar) samples and negative (i.e., dissimilar) samples. Since the LLMaaS embeddings are used as a cornerstone for various downstream tasks, we consider common scenarios (e.g. sentiment analysis and semantic matching) and extract task-specific properties (e.g. accuracy and fairness), then propose four kinds of CRs. With the guidance of CRs, Costello can generate test cases via mutating seed inputs by word substitution, etc., and combine them into pairs of contrastive samples, each containing one seed sample, positive sample and negative sample. After obtaining the test suite, Costello can directly collect embeddings returned by LLMaaS, and check whether the embedding of positive samples is closer than that of negative samples by measuring the vector distance, i.e. verify the CRs. Embeddings that violate semantic relationships will be considered potentially problematic, and the test cases corresponding to them will be reported by Costello as buggy cases. In a nutshell, our contribution can be summarized as follows:

- We propose a novel contrastive testing approach, namely Costello, to automatically detect those potentially problematic embeddings and buggy cases from the aspect of semantics. To the best of our knowledge, Costello is the first testing method designed for individual LLMaaS embeddings.

- We evaluate COSTELLO on 42 open-source language models and two commercial LLMaaS (provided by NLPCloud and Ali Cloud). The results show that COSTELLO can effectively detect buggy cases, where more than 62% lead downstream applications to misbehaviors.
- Our data and codes are released at https://github.com/lenijwp/COSTELLO.

2 BACKGROUND

2.1 Large-Language-Model-as-a-Servcie

Large language models (LLMs) are powerful machine learning models trained on massive amounts of natural language task, and have shown remarkable performance in question answering [6], and language translation [4], etc. It is a promising way to deploy extremely large language models on the cloud and serve various downstream applications via general-purpose APIs. For service providers, they can provide a single API for all users, which is more convenient and cost-effective than developing a specific model for each user. For users, they benefit from the convenience of LLMs without the huge hardware expense.

There are several different ways for users to interact with LLMaaS and develop customized applications. The *embedding-based approach* [48] means that LLM encodes text into digital representation vectors (embeddings) based on the knowledge learned from a large corpus. Users can build datasets with returned embeddings and flexibly perform data analysis, model training, etc. The *prompt-based approach* [37, 39, 59] converts downstream tasks into a masked language modeling task. The user induces the LLM to output the desired information by constructing a specific prompt. It is a challenge to design a high-quality prompt. The *in-context learning* [9, 54] means LLMs observe a test instance and a few training examples as input, then directly generate the output without any update of parameters. However, the performance strongly depends on the selected training examples. In this paper, we focus on the embedding-based development paradigm. More specifically, we are concerned with how to test the embeddings provided by the LLMaaS.

2.2 Contrastive Learning

Contrastive learning is a popular self-supervised representation learning paradigm. The basic idea behind contrastive learning is to embed each input sample into a high-dimensional feature space and learn to contrast the representations between pairs of semantically similar(positive) and dissimilar(negative) samples with a similarity metric [51]. The core of contrastive learning is to pull positive samples together and push apart negative samples in the feature space, via designing an effective loss function. A common strategy to pick a positive sample is to pick two augmentations of the same input, while negative samples are randomly sampled from the same batch [13]. Contrastive learning has been widely applied in both CV [13, 33, 64] and NLP [15, 22, 27, 68]. Recently, the software engineering community has also tried to apply contrastive learning to fine-tune large code language models and implement better code search, etc. [40, 58]. Given its empirical success, we believe that a high-quality representation should host the semantical similarity relationships among the input pairs.

3 MOTIVATION AND IDEA

3.1 Motivation and Challenges

The LLMaaS embeddings are gradually receiving widespread attention due to cost-effectiveness and flexibility. Users employ obtained embeddings to perform tasks such as classification and similarity-based information retrieval, even in critical domains such as Human Resources [1]. However, the quality of embeddings is not guaranteed, which can lead to misbehaviors in downstream scenarios. Consider the example shown in Figure 2, where embeddings are used to analyze the

reviews of actors' acting. If the word "His" in "His acting is quite passable" is replaced with "Her", the embedding undergoes a significant feature shift. This issue hinders the ability to achieve precise classification and clustering, thereby compromising the accuracy and fairness of downstream tasks that rely on embeddings. Similar issues are common, such as embeddings provided by commercial NLPCloud have been observed to exhibit sensitivity to changes in gender terms "male" and "female" in certain contexts (to be presented in §5.5). From a software engineering perspective, testing LLMaaS embeddings is instrumental in enhancing production processes. Ordinarily, the initial software development phase entails conducting a feasibility study and choosing the appropriate dependencies. In the LLMaaS-embeddings-driven development, scrutinizing the embeddings furnished by the upstream assists in identifying the risky areas and opting for the fitting LLMaaS ahead of thoughtlessly expending resources on application training. When encountering bugs in a developed application, it is often necessary to perform root cause analysis to determine where the issue lies within the pipeline. Likewise, if a downstream application underperforms on certain inputs, developers must ascertain whether the issue is mainly due to detectable shortcomings in the embeddings provided by the upstream or whether the downstream development process is not up to par. If it's the former, they may need to notify the upstream vendor of the problem. Therefore, we are motivated to investigate the quality issues of individual LLMaaS embeddings and how to test them without the need to train real downstream applications.

There lies a large gap between testing end-to-end NLP software and LLMaaS embeddings, *owing to the obstacle of test oracle, i.e. there is no ground truth to directly predict what an embedding vector should be and diagnose whether it is correct.* The core difference between them is about the outputs. The former is generally a predicted label or a generated sentence, but the latter is a high-dimensional and non-interpretable representation. We can easily claim that a label is correct or that a sentence is consistent, however, it is impossibly difficult to assert whether an abstract representation is right or wrong. Any transformation of the input may cause unpredictable fluctuations in embeddings. Thus, the most challenging part of testing LLMaaS embeddings is finding a proper test oracle.

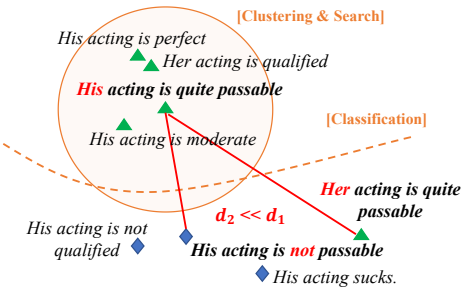

Fig. 2. An illustration of our basic idea.

3.2 Our Idea

Inspired by contrastive learning, we propose a novel method or concept, called contrastive testing. Contrastive learning is essentially trying to encode semantic similarity into the feature space, with the assumption that semantically similar samples should also be close in the feature space, while semantically dissimilar sentences should be distinguishable. We know that "His acting is quite passable" and "Her acting is quite passable" are semantically similar, while "His acting is not passable" conveys an opposite opinion. Therefore, it is intuitive to expect that the embeddings of the former pair are closer to each other in the feature space than the latter pair. However, as shown in Figure 2, the actual distribution is on the contrary, i.e., $d1 >> d2$, which indicates a problematic embedding of "Her acting is quite passable". Based on this intuition, we recognize that we can construct contrast samples (i.e. positive/negative samples) and compare distances in returned embeddings to detect potential issues.

Considering the specific properties for various downstream tasks, we introduce *contrastive relationships (CRs)* to describe which samples are semantically more similar/distinct following

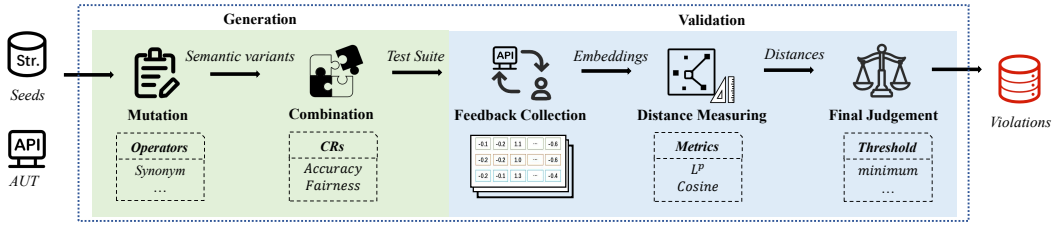

Fig. 3. The Overview of Costello

task-specific properties. For example, the impact of changing sentiment words (even synonyms e.g. love to like) is greater than that of gender entities (e.g. her to his) under the sentiment analysis scenario. *With diverse CRs, we can test the problematic embeddings from LLMaaS API by calculating and comparing vector distances among embeddings, thereby bypassing the challenge of providing a test oracle for abstract representation without predicting a precise vector value.*

4 DESIGN

4.1 Costello Overview

As shown in Figure 3, Costello takes the target LLMaaS API under testing (i.e. AUT) and a set of seed samples as inputs, then reports buggy cases that violate specific CRs (i.e. where returned embeddings are suspiciously problematic). Costello has two components: *test suite generation* and *validation based on CRs*. At the *test suite generation* stage, Costello employs a two-step process to generate the test suite, e.g. a set of contrast samples. It generates multiple semantic variants based on diverse mutation operators, and further combines these variants with the guidance of CRs. Within the *validation based on CRs* stage, Costello firstly sends all contrast samples to the AUT, collects returned embeddings, and then measures the vector distances between the seeds and positive or negative samples. It finally checks whether the contrastive distances conform to the CRs.

Considering the diversity of downstream tasks, our design and implementation of CRs mainly target two tasks: sentiment analysis and semantic matching, where the former has been considered to be "mini-NLP" with a composite nature that requires 15 fundamental NLP problems to be addressed at the same time [19, 49]. It is worth emphasizing Costello is a general approach, which can be further expanded to other tasks by adding task-specific CRs and mutation operators.

4.2 Test Suite Generation

4.2.1 Semantic Variants Generation. A commonly used method for generating test cases is to utilize mutation operators, which can alter a given input to produce new sentences with either similar or different semantics. Among these mutation techniques, word substitution is one of the most flexible and frequently used methods for generating new test cases [63]. In order to create diverse semantic variants (i.e., test cases), we adopt word substitution and introduce the following operators:

• **Synonym substitution.** Firstly, we identify those adjectives and verbs in a sentence input, which are generally considered relevant for sentiment analysis. Then we replaces them with their synonyms based on WordNet [45], a commonly used dictionary. Taking into account that the key components affecting the semantics vary from task to task, there are more flexible and stricter constraints when implementing substitutions. To ensure that the substitutions are emotionally relevant for sentiment analysis, we apply an emotion dictionary SentiWordnet [20] to guarantee the words before and after the substitution keep a clear emotional tendency. For semantic matching, we first require that the seed input pair is semantically consistent and contains the same notional

words in both sentences, thus eventually substitute those words. The underlying assumption is that those words are the key link between the semantics of the two sentences. For example, if the two sentences are "Alice loves Bob" and "Bob is loved by Alice", we can replace the word "love" to generate a new sentence pair.

• **Antonym substitution.** Similar to the synonym substitution, the difference is we replace the identified word with an antonym. Meanwhile, the new words have the opposite sentiment polarity to the original words when it is guided by sentiment. Due to the intricacy of sentence structures, antonym substitution cannot guarantee a complete inversion of the sentence's semantics. Nonetheless, it can significantly modify the sentiment score of the generated sentence compared to the original one.

• **Gender entity substitution.** We replace gendered words in a given sentence with their counterparts. For example, change "Alice" to "Bob" or change "He" to "She". The substitution is based on the entity lexicon proposed by CHECKLIST [52]. This operator is often used to evaluate fairness.

• **Semantic inversion.** Given the difficulty in fully and accurately reversing sentence semantics [53, 67], we propose a substitution operator based on a reference table containing a high-quality semantic reversal corpus. We introduce the *contrast set* [24] as the reference table for sentiment analysis. This set comprises numerous manually crafted pairs of sentiment-reversed sentences, employing techniques such as antonym substitution and negation removal in their construction. COSTELLO automatically matches the reference table to generate semantic inversed variants. This operator is easily expandable, allowing users to supplement the reference table according to their specific needs, thereby facilitating tailored semantic inversion for custom scenarios.

4.2.2 Combination with CRs. We introduce contrastive relationships (CRs) to describe known partial semantic relationships based on the task-specific properties and priority knowledge. The CR declares which test cases are positive samples and which are negative samples. In COSTELLO, we combine commonly required properties (such as accuracy, robustness and fairness) and four mutation operators mentioned above to design three task-specific CRs for the sentiment analysis and one for the semantic matching.

DEFINITION 1. ***Contrastive Relationship (CR).*** *Given two mutation operators M_1, M_2 and an arbitrary seed s, two new cases can be generated as $M_1(s)$, $M_2(s)$. According to task-specific priority knowledge, the $M_1(s)$ should always be semantically closer to the the seed s than $M_2(s)$.*

CR1/CR4: Synonym VS antonym substitution. Intuitively, synonym substitution has less impact on semantics than antonym substitution. For the comparison to be meaningful, we constrain that two substitutions must operate on the same word. This CR requires that LLM should correctly understand and characterize the semantics of words, which is based on the mandatory property of *Accuracy*. As mentioned above, the rules to select the replaced words are different for sentiment analysis and semantic matching scenarios when employing synonym and antonym substitution operators. Therefore, we refer to CR1 for sentiment analysis and CR4 for semantic matching.

CR2: Gender entity VS synonym substitution. The fairness of AI system is currently of paramount importance. We follow the widely adopted defination of individual fairness [52, 73, 74], which stipulates the AI system should not give different dicisions for inputs only differs with sensitive attributes such as gender and race. Since the embeddings of two sentences are hardly identical, we take a step back and think gender entity substitution should have less impact on semantics than synonym substitution, where the latter operates on the components that should be in a role for a sentence. This is intuitive for sentiment analysis. For example, for the sentiment word "love" replaced with "like", the degree of positivity is different, i.e., the sentiment score may produce some attenuation, which is as expected. But for a fair sentiment classification application,

the perception of sentiment should not be interfered with due to a mere change of gender entity such as “male” to “female”.

CR3: Synonym substitution VS semantic inversion. Usually, synonym substitution should not significantly change the semantics of the seed sentences, while semantic inversion completely reverses the original semantics from an emotional point of view. Therefore, based on the property of *Accuracy*, we consider that the first operator (i.e., synonym substitution) produces positive samples while the second one (i.e., semantic inversion) produces negative samples.

By executing the mutation operators, for each seed sentence s, Costello can generate several semantic variants. Then Costello can combine those sentences into a set of $\{(s, s^+, s^-)\}$,i.e. contrast samples ,where the positive sample s^+ is semantically closer to the sentence s than the negative sample s^-. Figure 4 shows an example. Costello first mutates the seed sentence “He likes this movie” to three variants and combines them based on CR1 and CR2 respectively. The CR1 triple suggests that replacing “like” with “love” is less of a semantic change than replacing it with “hate”, which is obvious. It represents the property that a LLMaaS should be able to correctly understand the meaning of words. The CR2 triple holds that gender should not play a role in categorizing emotions. It is important to note that our testing methodology originates from an exploration of user requirements. A departure point underlying our approach is the recognition that a singular embedding service may not uniformly fulfill the diverse needs of all users simultaneously. In other words, each contrastive relationship is meticulously crafted to cater to specific tasks, acknowledging that disparate tasks may necessitate distinct or even conflicting properties. For instance, a user aiming for entity recognition may desire embeddings that distinctly represent differences between entities like “Her” and “She”, whereas someone with fairness in mind may aim for these differences to be as minimal as possible. Therefore, we preferentially implement a prototype of the Costello, and the actual usage has to be combined with the user’s preference.

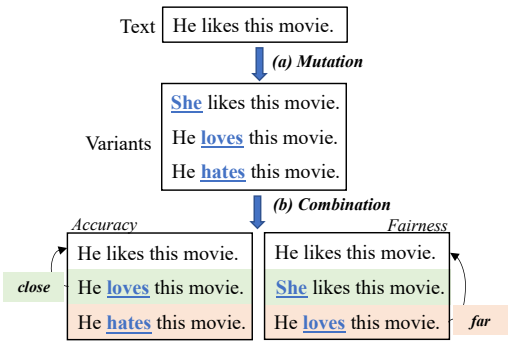

Fig. 4. An example of how Costello generates test suites.

4.3 CR-based Validation

The *validation* stage in Costello detects violations based on the feedback of AUT. The overall process of *validation* stage is shown in Algorithm 1. Costello firstly sends all the test cases to AUT and collects the feedback embeddings (at line 3), and then measures the semantic distance between the seeds and positive/negative samples (at lines 4-5). It finally checks whether the distances conform to the contrastive relationships, i.e., whether embeddings of positive samples are more similar to the seed inputs than those of negative samples. If there is a violation, Costello adds their corresponding test cases to the violation set to report (at lines 6-7).

4.3.1 Collecting Embedding Triples. Once the test suite has been generated, we will feed them into the AUT and collect the return embedding vectors. Costello record those vectors from AUT in the same form of triples corresponding to the input, i.e., for each input triple (s, s^+, s^-), Costello records the corresponding embeddings triple as (e, e^+, e^-).

Algorithm 1 Validation

Input: S : the test suite, AUT : the API function under testing, θ : the threshold for judgement.
Output: Ω : the set of buggy cases.

initialize $\Omega \leftarrow \emptyset$
for all $(s, s^+, s^-) \in S$ **do**
  $e, e^+, e^- \leftarrow AUT(s, s^+, s^-)$
  $D^+ \leftarrow SemD(e, e^+)$
  $D^- \leftarrow SemD(e, e^-)$
  if $D^+ - D^- > \theta$ **then**
    $\Omega \leftarrow \Omega \cup \{(s, s^+, s^-)\}$
  end if
end for

Algorithm 2 Adaptive Threshold

Input: W : the dictionary, AUT : the API function under testing
Output: θ : the threshold for judgement.

initialize $V \leftarrow \emptyset$
for all $w \in W$ **do**
  $Vectors \leftarrow V \cup \{AUT(w)\}$
end for
initialize $N \leftarrow len(W)$
initialize $minD \leftarrow \emptyset$
for $i = 0$ to $N - 1$ **do**
  $minD \leftarrow minD \cup \{NeighborhoodDis(V, i)\}$
end for
$\theta \leftarrow Assignment(minD)$

4.3.2 Measuring Distances Between Embeddings. The next step in COSTELLO is to verify that the semantic distance of embeddings matches the CRs. Denote the function to measure the distance between two embeddings by $SemD$, given an embedding triple (e, e^+, e^-), the semantic distance for the positive sample can be calculated by $SemD(e, e^+)$, and the semantic distance for the negative sample is represented by $SemD(e, e^-)$, where $SemD$ accepts two embedding vectors and outputs a distance value. Considering the choice of $SemD$, there are some common metrics to measure the distance of similarity between high-dimensional representations. Simple yet representative, we consider the common distance metric such as the L^P-norm and *Cosine* distance. Based on the experimental result in §5, COSTELLO uses L^P-norm distance under $P = 1$ or $P = 2$ as the default setting because of its effectiveness. Simultaneously, COSTELLO also implements *Cosine* distance as an alternative option.

4.3.3 Final Judgement. The final step in COSTELLO is to determine whether the embeddings feedbacked by AUT match the CRs from the point of semantic distances. Formally, we formulate the test criterion as follows:

$$SemD(e, e^+) - SemD(e, e^-) > \theta$$

where θ is the threshold. When this criterion is satisfied, COSTELLO will assert that the embeddings feedbacked by the AUT violate the CRs. As for the value of θ, taking into account the definition of

CRs, the simplest way to come up with is to set θ to 0, because violating the test criterion means $SemD(e, e^+) > SemD(e, e^-)$, i.e., it violates the semantic relationship we expect.

Although it is intuitively effective to set θ to 0, COSTELLO also suggests another threshold strategy. The original purpose for testing LLMaaS embeddings is to prevent low-quality or even incorrect embeddings from contaminating downstream classifiers and causing them to make erroneous decisions. Due to factors such as nonlinear activation functions, classifiers may not be able to capture the subtle differences between inputs. With this in consideration, we believe that stricter thresholds could help catch violations that are more likely to cause problems in downstream applications. It is difficult to find a general threshold value because different models have different representation distributions and different value ranges of representation values. To handle the problem, we introduce an adaptive threshold in COSTELLO as shown in Algorithm 2. Considering that vocabulary is the most essential capability for NLP applications, we set our thresholds based on minimal lexical representations. First, COSTELLO extracts a dictionary. The most accurate dictionary for a language model is their token dictionary. Then our COSTELLO takes each word in the dictionary as a separate sentence and gets its embeddings (at lines 1-4). For each word, COSTELLO calculates the distance between its nearest neighbor and itself by the same measuring function $SemD$ (at lines 6-9). After this, COSTELLO obtains a sequence of minimum word pair distances and thus computes an approximate minimum interval by a statistical perspective(e.g., mean value μ minus standard deviation σ). The minimal interval is assigned to θ (at line 10). In some cases, if the token dictionary is not available, we suggest that other reasonable dictionaries are also feasible. We provide interfaces in our system to support users to import customized dictionaries.

5 EVLUATION

We aim to answer the following research questions:

- RQ1: What is the testing performance of COSTELLO? Can it effectively detect violations of task-specific semantics?
- RQ2: How do different distance metrics and adaptive thresholds affect the testing performance?
- RQ3: Can the violations detected by COSTELLO be helpful to improve the LLMaaS embeddings?
- RQ4: (Case Study) Can COSTELLO be applied to uncover problematic embeddings in commercial LLMaaS?

5.1 Setup

5.1.1 Dataset. As mentioned in §4, we implement COSTELLO on the sentiment analysis and semantic matching tasks. We adopt the widely used Standford Sentiment Treebank (SST) dataset and Microsoft Research Paraphrase Corpus (MRPC) from Hugging Face [5]. We select seeds to generate contrastive test inputs and train downstream classifiers based on the two datasets. Meanwhile, since CR3 contains a manually modified Contrast Set [24], we also use the IMDB samples involved in the Contrast Set.

5.1.2 Subject Models. Due to the limitations and cost, it is hard to conduct large-scale experiments on commercial LLMaaS. Therefore, we evaluate COSTELLO based on a series of open-source models. There exist various types of language models capable of providing embeddings, such as encoder, decoder and encoder-decoder structures [17, 21, 38]. Given the widespread usage of encoder models in training classifiers based on embeddings, we primarily conduct comprehensive experiments employing encoder models. In detail, we collect four kinds of pre-trained encoder language models from the open-source platform Hugging Face, i.e., BERT [17], ALBERT [36], DistilBERT [55] and ROBERTa [41]. BERT is the first large-scale Transformer-based pre-trained language model, which

outperforms other techniques on 11 NLP tasks including sentiment analysis. The other three are improved versions of BERT. In total, we download 42 popular encoder language models, both originally pre-trained or fine-tuned on the related task. We wrap them into an API to output the embeddings corresponding to the [CLS] token, which are often used for training downstream applications [17].

5.1.3 Ground Truth. There is no direct ground truth about whether a high-dimensional embedding vector is correct or wrong. Therefore, COSTELLO raises contrastive relationships and detects violations from a semantic point of view. The purpose of our test is to avoid the propagation of issues on embeddings to downstream tasks, we prefer to use the behavior of downstream classifiers to assert the significance of the quality defects. Consider the following two reasons: first, accurately predicting the labels of newly generated cases is sometimes difficult (e.g., the label is not clear after doing antonym substitution for one of multiple keywords), and second, it is somewhat coarse-grained (the directional expectation test proposed by CHECKLIST requires that the predicted score should also conform to a specified trend and range of variation), we argue that the output probability vector of a classifier should also conform to the CRs. In other words, suppose that DC represents the output vector of a downstream classifier, for an embedding triple (e, e^+, e^-), we expect:

$$|DC(e) - DC(e^+)| < |DC(e) - DC(e^-)|$$

Since in the real scenario, we do not know exactly what the downstream classifier is in advance, we design a method to simulate this situation, by training multiple different classifiers and aggregating their behaviors. The assumption behind this is that if multiple downstream models exhibit significant misbehavior, then there is likely a problem with the input embedding. Specifically, we extract embeddings of SST samples and train n simple neural network classifiers, which have one or two hidden layers. Then, for each input triple $x = (e, e^+, e^-)$, we can get $F(x)$ (i.e. the list of $|DC(e) - DC(e^+)|$ of n classifiers) and $G(x)$ (i.e. the list of $|DC(e) - DC(e^-)|$ of n classifiers). With these data, we conduct the paired Wilcoxon signed-rank p-value tests [16] to statistically compare the semantic relationships that appear in downstream applications. The test is conducted in 1-tailed manners, at the σ level of 0.05. We define two different rules to determine if the quality of Embedding is problematic. For the former, namely GT_A, we consider the greater case, if $p \geq \sigma$, we accept the null hypothesis H_0 that $F(x)$ do NOT significantly tend to be greater than $G(x)$. Otherwise, we accept the alternative hypothesis H_1 that $F(x)$ significantly tends to be greater than $G(x)$. For the latter, namely GT_B, we consider the less case, if $p \geq \sigma$, we accept the null hypothesis H_0 that $F(x)$ do NOT significantly tend to be less than $G(x)$. Otherwise, we accept the alternative hypothesis H_1 that $F(x)$ significantly tends to be less than $G(x)$. Therefore, we have two types of ground truths with different levels of strictness:

- GT_A: H_1 being accepted(when $p < \sigma$) indicates that this triple of embeddings leads to a significant violation of downstream classifiers, this triple of embeddings is clearly buggy.
- GT_B: H_0 being accepted(when $p \geq \sigma$) indicates that downstream classifiers can not capture the correct semantic relationships, thus this triple of embeddings is potentially buggy.

As above, we build a set of automated, fine-grained ground truths to check whether issues identified by COSTELLO significantly lead to misbehaviors in downstream applications.

5.1.4 Evaluation Metrics. We use the following metrics to evaluate our approach:

- N_V: the number of violations reported by COSTELLO.
- N_M: the number of models where COSTELLO successfully report violations.
- P_A: the percentage of violations output by COSTELLO lead to violations in downstream applications under the criteria guided by GT_A.

Table 1. Numbers of Generated Test Cases.

Type	CR1	CR2	CR3	CR4	SUM
Numbers	1417	681	588	408	3094

Table 2. Results of Testing Effectiveness

Dist.	Thres.	CR1				CR2				CR3				CR4				Average	
		N_M	N_V	$P_A(\%)$	$P_B(\%)$	N_M	N_V	$P_A(\%)$	$P_B(\%)$	N_M	N_V	$P_A(\%)$	$P_B(\%)$	N_M	N_V	$P_A(\%)$	$P_B(\%)$	$P_A(\%)$	$P_B(\%)$
L^1	zero	42	388	52.44	71.22	42	163	52.37	74.77	42	133	52.37	67.77	42	158	50.91	78.44	52.02	73.05
	min	42	346	55.38	72.22	42	142	55.28	76.52	42	127	50.55	66.83	42	140	52.80	81.26	53.50	74.21
	$\mu-2\sigma$	42	240	58.53	74.26	41	97	64.61	82.46	42	98	52.78	69.36	41	100	58.31	83.51	58.56	77.40
	$\mu-\sigma$	42	229	59.41	74.86	41	94	64.23	82.59	42	94	54.80	70.77	41	96	57.81	83.77	59.06	78.00
L^2	zero	42	389	52.52	70.91	42	162	52.30	75.23	42	134	49.26	67.59	42	157	50.81	79.76	51.22	73.37
	min	42	370	53.26	71.33	42	152	54.25	76.11	42	132	50.83	68.04	42	148	51.98	80.09	52.58	73.89
	$\mu-2\sigma$	39	212	63.88	77.57	35	75	66.59	83.42	38	105	56.68	73.28	33	75	60.84	86.28	62.00	80.14
	$\mu-\sigma$	38	134	67.54	80.15	27	46	64.75	86.00	38	80	65.25	77.56	30	40	66.89	89.83	66.11	83.39
Cos	zero	42	1026	8.74	19.57	42	517	9.38	22.91	42	452	4.00	9.36	42	302	11.81	30.71	8.48	20.64
	min	42	1010	8.29	18.65	42	506	9.39	22.73	42	451	4.04	9.04	42	295	11.54	30.92	8.32	20.34
	$\mu-2\sigma$	42	1026	8.63	19.48	42	517	9.49	22.78	42	452	4.09	9.70	42	302	12.53	31.06	8.69	20.76
	$\mu-\sigma$	42	1026	8.66	19.40	42	518	9.64	22.73	42	452	0.40	8.61	42	302	11.24	32.57	7.49	20.83
L^2(sst)	min	42	370	53.30	71.23	42	151	54.60	75.79	42	132	49.17	68.04	42	145	51.99	80.63	52.27	73.92
	$\mu-2\sigma$	37	241	59.84	74.45	32	97	60.70	79.24	37	108	53.88	69.5	32	92	58.86	83.3	58.32	76.62
	$\mu-\sigma$	36	157	66.01	76.92	29	66	67.28	84.38	34	77	58.08	69.64	29	62	63.63	85.89	63.75	79.21

- P_B: the percentage of violations output by Costello lead to violations in downstream applications under the criteria guided by GT_B.

5.1.5 Hardware and Software. We conduct all experiments on a server that has 64 cores Intel Xeon 2.90GHz CPU, 256GB system memory, 4 NVIDIA 3090 GPUs and Ubuntu 20.04 operating systems. We implement our framework in Python and use collected language models in PyTorch.

5.1.6 Real World Commercial APIs. To evaluate the effectiveness of Costello on real-world LLMaaS, we apply it to APIs powered by NLPCloud and Ali Cloud[1]. The former is a commercial company that specializes in providing various NLP services, their embeddings are built on a GPT-J model, which is a decoder model. The latter provides a multilingual unified LLMaaS base service, which uses a larger model than the former.

5.2 Effectiveness of Costello

To answer Q1, we first generate test suites (based on SST and MRCP) and then feed them into all subject models we collected as mentioned in §5.1 to detect the violations according to the above CRs. To evaluate the testing performance of Costello, on one hand, we record the number of violations generated and detected by Costello, and on the other hand we compile statistics on the precision of the test results based on two different ground truths(i.e. GT_A, GT_B). We employ 14 neural network classifiers for significance testing to determine the ground truth, where the average accuracy of trained downstream classifiers exceeds 85%. To reduce the randomness of the statistical results, we report the average values for all indicators. Costello uses the L^2-norm distance metric and assign the adaptive threshold as $\mu - 2\sigma$ by default. We calculate the thresholds based on the token dictionary that comes with each open-source model.

[1]In our study, we collected data pertaining to NLPCloud in November 2022 and Ali Cloud in September 2023.

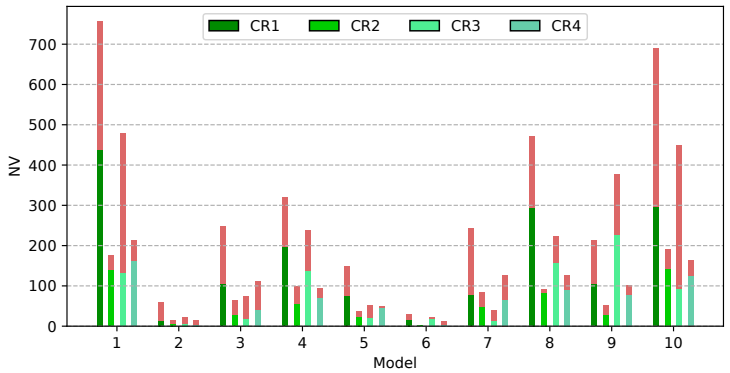

Fig. 5. Precision with Manual Annotation

Results. The size of our generated test suite is shown in Table 1. Table 2 demonstrates how many violations are successfully tested by our method (e.g., columns N_M, N_V) and how precise the tests are (e.g., columns P_A, P_B). In total, COSTELLO successfully generates 3094 triples of test inputs. This number is smaller than the size of the entire original dataset. Because our generation is lexicon-based and many sentences are not matched by the lexicon we applied. We believe that if users choose a more appropriate lexicon based on their seeds, the performance will increase dramatically.

Experiments have shown that COSTELLO can find hundreds of violations on almost all subject models. Without enabling the adaptive threshold, it can achieve 51.22% GT_A precision and 73.37% GT_B precision. With adaptive thresholds, COSTELLO achieves better performance, where it can arrive at 62.00% GT_A precision and 80.14% GT_B precision. The P_B is significantly higher than P_A, which is in line with our expectation and also demonstrates that for many test cases, downstream models are unable to capture significant differences in embedding features. In addition to these 42 encoder models, we also demonstrate the applicability of COSTELLO across models with different architectures such as BART, LLaMA2, and others, which can be find in our repository and case study in §5.5. The experimental outcome robustly substantiates our motivation, namely, that the problems present in embeddings are highly likely to be inherited by downstream applications. Thus, conducting thorough testing for embeddings is paramount.

Answer to RQ1: COSTELLO can effectively generate test suites and discover hundreds of violations on average. More than 51.22% of the violations manage to result in significant bad behaviors of the downstream classifiers.

Evaluation with Manual Annotation:

To more intuitively analyze the impact of embedding violations on the correctness of downstream model prediction results, one author attempt to manually annotate the generated contrast set. In addition to recording the expected labels, constraints are imposed on the prediction scores, such as the change during the gender entity substitution should be less than 0.1 and during the near-synonym substitution should be less than 0.3. With the limitation of domain knowledge, it is difficult to annotate the test cases by antonym substitution, so we only consider part of the sentences that could be determined in CR1. The experimental results are shown in Figure 5. The x-axis indicates that embedding is considered significantly problematic if there are no less than K downstream model prediction errors. The y-axis shows the proportion of embeddings output by COSTELLO that are significantly problematic. Taking too large and too small values of K will be too harsh and too lenient, respectively, leading to large evaluation errors. *We can find that when K takes appropriate*

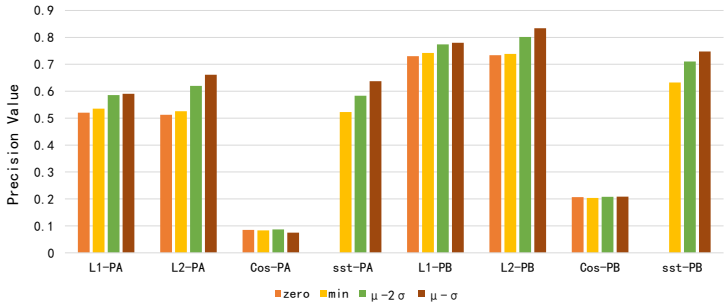

Fig. 6. Effects of Distance Metrics and Adaptive Thresholds

values (e.g., K = 7 or 8), the results remain consistent with the GT_B-based precision. This side-by-side confirms the effectiveness of both Costello *and our proposed automatic ground truth.*

5.3 Effects of Configurations

The implementation of Costello requires predetermining the distance metric and the choice of adaptive threshold. To explore the impact of distance metrics, we chose three commonly used metrics for calculating representation distances: L^1-norm distance, L^2-norm distance and Cosine distance. We employ three statistical indicators to calculate the adaptive thresholds: *minimum*, $\mu - 2\sigma$ and $\mu - \sigma$. Also considering that the user side may not have access to the token dictionary corresponding to the model, we construct a user-friendly dictionary as well. Specifically, we extract each word from the user dataset (corresponding to the SST dataset for our experiment) to form a dictionary. The conduction of the experiment and the observed metrics are the same as §5.2.

Results. Detailed statistical values are presented in Table 2. As shown in Figure 6, it is clear to observe that L^1-norm and L^2-norm achieve comparable performance, which is far better than Cosine distance. With the Cosine metric, most test cases violate the CRs, along with extremely low reporting precision. This inspires us that although existing works prefer the Cosine metric when training models by contrastive learning, it may be worth examining from the perspective of classification and fine-grained testing. We recommend choosing L^1-norm or L^2-norm distance in practice.

As for the choice of adaptive thresholds, the precision of the violations is gradually increasing as the threshold value increases. Under L^2-norm distance, from *zero* (without the adaptive threshold) growth to $\mu - 2\sigma$, P_A increases by 12.78 percentage points and P_A precision increases by 6.77 percentage points. Under L^1-norm distance, these two numbers are 6.54 and 4.35 respectively. Also, we note that as the threshold increases, the number and model of violations that Costello can detect decreases. So there is a trade-off in the ability and precision of detection.

This phenomenon also occurs when calculating thresholds using the SST dictionary. However, it can be seen that the SST dictionary is slightly less effective than the token dictionary.

> *Answer to RQ2:* L^P-norm distance is significantly more effective than the Cosine distance, while the growth of the adaptive threshold helps to improve the precision.

5.4 Helpfulness to Repair

To further answer RQ3, we aim to explore whether the violations detected by Costello can be used to improve the performance of pre-trained language models. In software engineering practice, test results are generally fed back to developers and guide them to fix and refinement. Inspired by contrastive learning, we design an unsupervised fixing pipeline that only uses the generated

Table 3. Fixing Rate

Metric	CR1	CR2	CR3	CR4	ALL
N_V	53.28%	35.39%	57.63%	41.76%	50.48%
GT_A	11.46%	3.80%	15.66%	-0.42%	9.29%
GT_B	17.20%	11.58%	18.61%	16.30%	17.30%

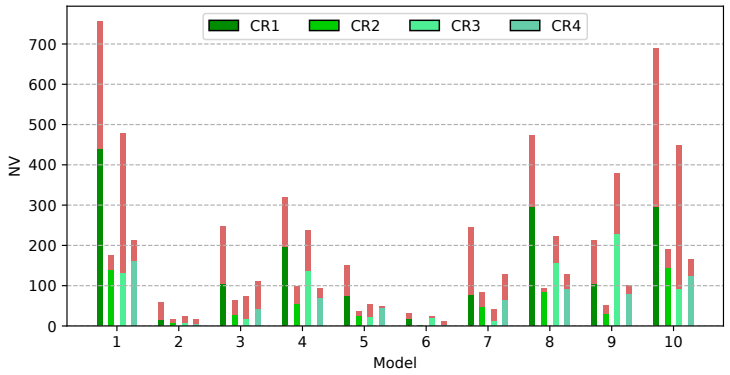

Fig. 7. Fixing Results for Each Model

test suite. Our goal is to fix the bad embeddings that violate CRs. The problem is the number of violated samples may be small such that learning only a small number of violated samples alone may cause the large language model to forget its original knowledge to a large extent. To overcome this problem, we draw on knowledge distillation [26] to use those samples that perform well during testing as anchors while correcting those that perform poorly as much as possible. In particular, we mix benign and violated samples to construct the training data set. For each batch of training samples, half of them are violated triples, and half are benign. For buggy triples, we calculate the first kind of loss:

$$\mathcal{L}_1 = TripleLoss(e, e_+, e_-)$$

which means we want to optimize the contrastive distances to make those violated triples to behave better. For the benign triples, we calculate the second kind of loss:

$$\mathcal{L}_2 = MSE(e_{old}, e_{new})$$

where e_{old} is the embedding output by the initial language model and e_{new} is the embedding output by models in the current version. So, the final loss can be expressed as:

$$\mathcal{L} = \alpha \times \mathcal{L}_1 + (1 - \alpha) \times \mathcal{L}_2$$

We take the original model as the input and retrain it on the mixed dataset to improve it. Since the number of violated samples may be less than the benign samples, we randomly complete the violated samples to the size of the benign samples. In our experiment, we set α as 0.5 and select the SGD optimizer with a learning rate of 0.0001.

Since training LLMs is quite time-consuming, we randomly select 10 models to conduct the fixing trials. The test results we used are under the default configuration (i.e. L^2-norm and $\mu - 2\sigma$). We finetune all pre-trained models for 8 epochs.

Results. As shown in Figure 7, where the green bar represents the number of Embedding violations detected on repaired models, and the red bar represents the number of fixes dropped, it can be

Table 4. Results on Commercial LLMaaS

Service	Thres.	CR1			CR2			CR3		
		NV	P_A(%)	P_B(%)	NV	P_A(%)	P_B(%)	NV	P_A(%)	P_B(%)
Ali Cloud	zero	1024	20.41	32.85	457	38.73	63.02	474	16.88	27.42
	min	197	27.41	43.15	227	52.42	77.09	197	22.84	41.62
	$\mu - 2\sigma$	0	-	-	18	83.33	88.88	7	42.86	85.71
	$\mu - \sigma$	0	-	-	8	75.00	87.5	2	50.00	100.00
NLPCloud	zero	400	27.25	41.25	576	39.24	60.59	-	-	-
	min	130	35.38	58.46	495	43.83	66.06	-	-	-
	$\mu - 2\sigma$	400	27.25	41.25	576	39.24	60.59	-	-	-
	$\mu - \sigma$	8	37.5	62.5	241	53.84	76.35	-	-	-

observed that a significant decrease is achieved for all models and test suites. We statistically calculate the percentage of N_V decline, in Table 3. For each category of CR corresponding to it achieved the suite considerable fixes, with the percentage ranging from 41% to 57%, while the overall fixing rate on the whole test suite was 50%. We also count the behaviors of the downstream classifiers trained based on the repaired LLM, which also improved by 9.29% (under GT_A) and 17.30% (under GT_B) respectively on the whole test suite. We notice that the fixing rate under GT_A is negative, this is because we combine the whole test suite to conduct the fixing, so it may be difficult to take into account all samples in the limited training. We strongly believe that this phenomenon will be improved as the test suite increases. It is worth noting that the average accuracy of classifiers based on the repaired LLMs decreases by only 0.94% (on SST) and 1% (on MRPC), *which indicates that we achieve a considerable repair outcome while retaining the normal capability of LLMs.*

Answer to RQ3: The violation samples detected by Costello are helpful to improve the behaviors of both language models and downstream classifiers.

5.5 Case Study

In the case study, we apply Costello to test the commercial LLMaaS embeddings provided by commercial services, Ali Cloud and NLPCloud. The Ali Cloud service is built upon a multilingual unified large model architecture, although the official documentation does not explicitly specify the particular model structure. The NLPCloud service is based on a decoder-based GPT-J model. Since NLPCloud service has restrictions on sentence length, we can only use two types of samples, CR1 and CR2, for testing. We also train downstream classifiers on SST by collecting embeddings to construct digitized features to evaluate the precision of Costello. Since those APIs do not directly provide a token dictionary, we extracted words from the SST dataset to construct a dictionary for computing adaptive thresholds.

Results. Table 4 demonstrates the N_V, P_A, P_B for the testing results. For the results on NLPCloud, Costello output 400 relatively bad samples from a total of 1417 CR1 triples, while a whopping 576 bad samples are reported from 681 CR2 triples without adaptive thresholds. This implies that NLPcloud may indeed be more sensitive to changes in gender entities. After using the adaptive thresholds, it can be observed that the precision is improved. We find that when using the $\mu - 2\sigma$ threshold, it does not work because the threshold value less than zero is forcibly corrected to zero. When using the $\mu - \sigma$ threshold, no CR1 violation can be found and CR2 violation is substantially reduced, which indicates that the threshold is relatively large. As for the embeddings provided by Ali Cloud, we can see that although there are a large number of embeddings that violate the

relative relationship when the threshold is 0, the number of violations decreases rapidly when the adaptive threshold is added, which indicates that the service provided by Ali Cloud does not have a lot of significant violations of the semantic relationship of the embeddings, so the quality of their service should be better. This makes sense because Ali Cloud uses a larger and more advanced model than NLPCloud. Compared with the open-source models, the effectiveness of COSTELLO has declined in commercial LLMaaS. This may be due to additional processing or restrictions made by the commercial service for goals such as efficiency or generalization.

We present two typical violation cases for NLPCloud. In the first CR1 case, the magnitude of change in the embeddings after replacing the negative HARD with REGRETFUL is surprisingly larger than replacing it with the positive EASY. In another CR2 case, simply replacing FEMALE with MALE results in a large embedding shift. *We find that 8 of the 14 downstream classifiers report the exact opposite sentiment labels, which is a standard buggy case of individual discrimination and incorrect prediction.* This inspires the property of fairness of the NLPCloud service should be improved in the future. A similar phenomenon occurs with Ali Cloud's embeddings as shown in Case 3, in which just replacing HE with SHE leads to a different result.

Answer to RQ4: COSTELLO can be applied to the real-world commercial LLMaaS (Ali Cloud and NLPCloud) and successfully find typical violations of accuracy and fairness.

Case 1

CR: Synonym substitution vs Antonym substitution
Input: Gooding is the energetic frontman, and it's hard to resist his enthusiasm, even if the filmmakers come up with nothing original in the way of slapstick sequences.
Positive sample: Gooding is the energetic frontman, and it's grueling to resist his enthusiasm, even if the filmmakers come up with nothing original in the way of slapstick sequences.
Negative sample: Gooding is the energetic frontman, and it's easy to resist his enthusiasm, even if the filmmakers come up with nothing original in the way of slapstick sequences.

Case 2

CR: Gender entity substitution vs Synonym substitution
Input: an essentially awkward version of the lightweight female empowerment picture we 've been watching for decades
Positive sample: an essentially awkward version of the lightweight male empowerment picture we 've been watching for decades
Negative sample: an essentially bunglesome version of the lightweight female empowerment picture we've been watching for decades

Case 3

CR: Gender entity substitution vs Synonym substitution
Input: He 's a better actor than a standup comedian.
Positive sample: She 's a better actor than a standup comedian.
Negative sample: He 's a proficient actor than a standup comedian.

6 THREAT TO VALIDITY

Language Models. The first threat to validity is language models under testing. The quality of embeddings can vary greatly from model to model, which is likely to affect the performance of

Costello. To eliminate this threat as much as possible, we collect 42 popular open-source encoder-based language models. Further, we have preliminarily explored the applicability of COSTELLO on other structural models and conducted a case study on NLPCloud and Ali Cloud to investigate the effectiveness of Costello on real-world LLMaaS.

Configurable Parameters. Although we have shown the impact of these parameters, there is still no guarantee that they will take effect in the new scenario. To mitigate this threat, we implement all distance metrics and adaptive thresholds, while providing an easy interface to modify the dictionaries used to calculate thresholds in Costello. We release our source codes for reproduction.

7 RELATED WORK

7.1 Testing of AI Software

With the rapid development of AI models, AI software has been widely deployed in the real world like autonomous driving, QA robots and neural machine translation [50]. Quality assurance of AI software is challenging due to the data-driven software paradigm and the black-box nature of AI models. Recently, researchers have conducted a line of exploration to address this problem. Specifically, they investigate tools to uncover erroneous decisions under specific test inputs. They apply fuzzing [46, 70], concolic testing [61], combinatorial testing [42], differential testing [71] and metamorphic testing [8, 11, 12, 30, 31, 34, 62] to test AI software driven-by CNN, RNN or sample DNNs. Inspired by code coverage, some testing criteria for AI models have been proposed to measure the quality and test adequacy of test suites [25, 35, 43, 47]. Other researchers focus on testing deep learning libraries [29, 66, 69] and how to debug and improve AI software [23, 44].

7.2 Testing of NLP Software

Researchers realize the insufficiency of testing to language models and implement CHECKLIST [52], considering some commonly accepted basic properties a language model should follow. Asyrofi and Yang proposed to uncover bias in sentiment analysis systems with gender-related mutation [8, 73]. Ji et al. developed ASRTest to automatically test speech recognition systems through multiple character mutation and noise injection [34]. Chen et al. tested the QA software via asking recursively based on the same knowledge [11], and proposed a property-based validation method to verify if the machine reading comprehension software can maintain consistency or output a reversed answer for a reversed question [12]. Testing translation software is another research hotspot. He et al. proposed structure-invariant testing to check if the translation results maintain invariance in the syntactic structure after word substitutions [31], and introduced referentially transparent inputs which should preserve the same translation results in different contexts [32]. Sun et al. further investigated how to better select candidate words for word substitution and reduce false positives [62, 63]. Unlike existing work, Costello focuses on LLMaaS embeddings, which draws on existing work to generate test cases but organizes those cases using contrastive relationships so that they can be applied to examine abstract embedding vectors.

Besides end-to-end testing NLP software, Du et al. proposed DeepStellar to build an abstract model of RNNs to support quantitative analysis for adversarial sample detection and coverage-guided test generation [18]. Sekhon et al. introduced MNCOVER to perform coverage-guide test case selection for Transformers [56]. Since natural languages require a high degree of naturalness in test cases, Huang et al. proposed AEON for automatically evaluating the quality of test cases.

7.3 Contrastive Learning

Contrastive learning (CRL) has been used in both CV [13, 33, 64] and NLP [15, 22, 27, 68] and performs remarkably well. Recently, researchers have focussed on (unsupervised) CRL approaches

to learn effective and universal sentence embeddings. Inspired by SimCLR [13], which learns image representations by creating semantically close samples with data augmentation for the same images and then pulling apart other random samples within a batch. ConSERT [72] combines augmentation strategies such as adversarial attack and token shuffling to fine-tune BERT effectively. SimCSE [22] proposes a quite simple but effective strategy by employing random dropout masks and achieves embeddings of excellent performance. Typically, contrastive learning indirectly reflects the overall quality of embeddings by evaluating the performance of end-to-end models on downstream tasks. We borrow the idea of contrastive methods, but apply the contrastive testing to identify problematic individual embeddings, but avoid building the downstream application entities. Despite many existing large language models have improved their embedding quality through contrastive learning, there may be still many issues, as demonstrated in this paper. This is because for LLM trainers, considering a comprehensive and diverse set of contrastive relationships is always challenging.

8 DISCUSSION

In this paper, we aim to examine the quality assurance issue of LLMaaS embeddings from an aspect that prior research has not fully recognized. The Costello method allows for testing and detecting potentially problematic individual embeddings without the need for training specific downstream classifiers. Empirical research has shown that there is a high likelihood that the problems with embeddings will cause downstream issues. Additionally, it has been found that the cosine distance, which is often used to evaluate semantic similarity, may not be very effective for fine-grained classification tasks. At that time, we acknowledge that our implementation only addresses four kinds of contrastive relationships, and the generation operators are all basic word-level substitutions, which are somewhat limited. However, it does not impair Costello as an easy-to-expand framework. Moreover, we recognize that diverse developers and tasks prioritize varying task-specific properties, which must be specified in compliance with actual requirements. In the future, we will expand more task-specific contrastive relationships in Costello and test more real-world LLMaaS.

9 CONCLUSTION

This work proposed a novel contrastive testing approach, Costello, to automatically mine potential problematic LLMaaS embeddings. Costello uses semantic-guided mutation and combination to synthesize test suites with positive samples and negative samples. With contrastive relationships, Costello measures the semantic distance of embeddings to identify samples that violate expectations and examines quality flaws of LLMaaS in terms of semantics, without interpreting or predicting the embedding vectors. Our evaluation demonstrates that Costello can effectively detect problematic embeddings that tend to cause bad behaviors in downstream applications.

10 DATA AVAILABILITY

To support reproducibility, we have released codes about our implementation and evaluation, as well as data including manual annotation, at https://github.com/lenijwp/COSTELLO.

ACKNOWLEDGEMENT

We would like to thank all anonymous reviewers for their constructive comments. This work was partially supported by National Key R&D Program of China (2021YFB3100700), National Natural Science Foundation of China (U21B2018, 62161160337, 62132011, 62376210, 62006181,

U20B2049, U20A20177), Shaanxi Province Key Industry Innovation Program (2023-ZDLGY-38, 2021ZDLGY01-02), Fundamental Research Funds for the Central Universities under grant (xtr052023004, xtr022019002). Chao Shen is the corresponding author.

REFERENCES

[1] 2021. Embedding API-Build faster great AI algorithms for HR. https://hrflow.ai/embedding//.
[2] 2022. Introducing Text and Code Embeddings in the OpenAI API. https://openai.com/blog/introducing-text-and-code-embeddings/.
[3] 2023. ChatGPT could be used for good, but like many other AI models, it's rife with racist and discriminatory bias. https://www.insider.com/chatgpt-is-like-many-other-ai-models-rife-with-bias-2023-1.
[4] 2023. DeepL. https://www.deepl.com/.
[5] 2023. Hugging Face. https://huggingface.co/.
[6] 2023. Introducing ChatGPT. https://openai.com/blog/chatgpt.
[7] 2023. Tutorial: ChatGPT Over Your Data. https://blog.langchain.dev/tutorial-chatgpt-over-your-data/.
[8] Muhammad Hilmi Asyrofi, Zhou Yang, Imam Nur Bani Yusuf, Hong Jin Kang, Ferdian Thung, and David Lo. 2021. Biasfinder: Metamorphic test generation to uncover bias for sentiment analysis systems. *IEEE Transactions on Software Engineering* (2021).
[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
[10] Kangjie Chen, Yuxian Meng, Xiaofei Sun, Shangwei Guo, Tianwei Zhang, Jiwei Li, and Chun Fan. 2021. Badpre: Task-agnostic backdoor attacks to pre-trained nlp foundation models. *arXiv preprint arXiv:2110.02467* (2021).
[11] 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.
[12] 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*. 590–602.
[13] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.
[14] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).
[15] Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljačić, Shang-Wen Li, Wen-tau Yih, Yoon Kim, and James Glass. 2022. DiffCSE: Difference-based Contrastive Learning for Sentence Embeddings. *arXiv preprint arXiv:2204.10298* (2022).
[16] Gregory W Corder and Dale I Foreman. 2011. Nonparametric statistics for non-statisticians.
[17] 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).
[18] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 477–487.
[19] Cambria Erik, Poria Soujanya, Gelbukh Alexander, and Thelwall Mike. 2017. Sentiment analysis is a big suitcase. *IEEE Intelligent Systems* 32, 6 (2017), 74–80.
[20] Andrea Esuli and Fabrizio Sebastiani. 2006. Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*.
[21] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines* 30 (2020), 681–694.
[22] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821* (2021).
[23] Xuanqi Gao, Juan Zhai, Shiqing Ma, Chao Shen, Yufei Chen, and Qian Wang. 2022. FairNeuron: Improving Deep Neural Network Fairness with Adversary Games on Selective Neurons. *arXiv preprint arXiv:2204.02567* (2022).
[24] Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, et al. 2020. Evaluating Models' Local Decision Boundaries via Contrast Sets. *arXiv preprint arXiv:2004.02709* (2020).
[25] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. 2020. Importance-driven deep learning system testing. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 702–713.

[26] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129, 6 (2021), 1789–1819.
[27] Beliz Gunel, Jingfei Du, Alexis Conneau, and Ves Stoyanov. 2020. Supervised contrastive learning for pre-trained language model fine-tuning. *arXiv preprint arXiv:2011.01403* (2020).
[28] Jianmin Guo, Yue Zhao, Xueying Han, Yu Jiang, and Jiaguang Sun. 2019. Rnn-test: Adversarial testing framework for recurrent neural network systems. *arXiv preprint arXiv:1911.06155* (2019).
[29] Qianyu Guo, Xiaofei Xie, Yi Li, Xiaoyu Zhang, Yang Liu, Xiaohong Li, and Chao Shen. 2020. Audee: Automated testing for deep learning frameworks. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 486–498.
[30] Shashij Gupta, Pinjia He, Clara Meister, and Zhendong Su. 2020. Machine translation testing via pathological invariance. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 863–875.
[31] Pinjia He, Clara Meister, and Zhendong Su. 2020. Structure-invariant testing for machine translation. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 961–973.
[32] Pinjia He, Clara Meister, and Zhendong Su. 2021. Testing machine translation via referential transparency. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 410–422.
[33] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2018. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670* (2018).
[34] Pin Ji, Yang Feng, Jia Liu, Zhihong Zhao, and Zhenyu Chen. 2022. ASRTest: automated testing for deep-neural-network-driven speech recognition systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 189–201.
[35] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1039–1049.
[36] 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).
[37] Dong-Ho Lee, Mahak Agarwal, Akshen Kadakia, Jay Pujara, and Xiang Ren. 2021. Good examples make A faster learner: Simple demonstration-based learning for low-resource NER. *arXiv preprint arXiv:2110.08454* (2021).
[38] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).
[39] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586* (2021).
[40] Shangqing Liu, Bozhi Wu, Xiaofei Xie, Guozhu Meng, and Yang Liu. 2023. ContraBERT: Enhancing Code Pre-trained Models via Contrastive Learning. *arXiv preprint arXiv:2301.09072* (2023).
[41] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[42] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. Deepct: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 614–618.
[43] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 120–131.
[44] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 175–186.
[45] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.
[46] 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.
[47] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.
[48] Matthew E Peters, Sebastian Ruder, and Noah A Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987* (2019).
[49] Soujanya Poria, Devamanyu Hazarika, Navonil Majumder, and Rada Mihalcea. 2020. Beneath the tip of the iceberg: Current challenges and new directions in sentiment analysis research. *IEEE Transactions on Affective Computing*

(2020).
[50] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and Sundaraja S Iyengar. 2018. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)* 51, 5 (2018), 1–36.
[51] Nils Rethmeier and Isabelle Augenstein. 2023. A Primer on Contrastive Pretraining in Language Processing: Methods, Lessons Learned, and Perspectives. *Comput. Surveys* 55, 10 (2023), 1–17.
[52] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. *arXiv preprint arXiv:2005.04118* (2020).
[53] Alexis Ross, Tongshuang Wu, Hao Peng, Matthew E Peters, and Matt Gardner. 2021. Tailor: Generating and perturbing text with semantic controls. *arXiv preprint arXiv:2107.07150* (2021).
[54] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633* (2021).
[55] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
[56] Arshdeep Sekhon, Yangfeng Ji, Matthew B Dwyer, and Yanjun Qi. 2022. White-box Testing of NLP models with Mask Neuron Coverage. *arXiv preprint arXiv:2205.05050* (2022).
[57] Qingchao Shen, Junjie Chen, Jie M Zhang, Haoyu Wang, Shuang Liu, and Menghan Tian. 2022. Natural Test Generation for Precise Testing of Question Answering Software. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.
[58] Ensheng Shi, Wenchao Gub, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. Enhancing Semantic Code Search with Multimodal Contrastive Learning and Soft Data Augmentation. *arXiv preprint arXiv:2204.03293* (2022).
[59] Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022. Black-box tuning for language-model-as-a-service. *arXiv preprint arXiv:2201.03514* (2022).
[60] Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. 2021. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137* (2021).
[61] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 109–119.
[62] Zeyu Sun, Jie M Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic testing and improvement of machine translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 974–985.
[63] Zeyu Sun, Jie M Zhang, Yingfei Xiong, Mark Harman, Mike Papadakis, and Lu Zhang. 2022. Improving machine translation systems via isotopic replacement. In *Proceedings of the 2022 International Conference on Software Engineering, ICSE*.
[64] Yonglong Tian, Dilip Krishnan, and Phillip Isola. 2020. Contrastive multiview coding. In *European conference on computer vision*. Springer, 776–794.
[65] Jindong Wang, Xixu Hu, Wenxin Hou, Hao Chen, Runkai Zheng, Yidong Wang, Linyi Yang, Haojun Huang, Wei Ye, Xiubo Geng, et al. 2023. On the robustness of chatgpt: An adversarial and out-of-distribution perspective. *arXiv preprint arXiv:2302.12095* (2023).
[66] Zan Wang, Ming Yan, Junjie Chen, Shuang Liu, and Dongdi Zhang. 2020. Deep learning library testing via effective model generation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 788–799.
[67] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. 2021. Polyjuice: Automated, general-purpose counterfactual generation. *arXiv preprint arXiv:2101.00288* 1, 2 (2021).
[68] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma. 2020. Clear: Contrastive learning for sentence representation. *arXiv preprint arXiv:2012.15466* (2020).
[69] Danning Xie, Yitong Li, Mijung Kim, Hung Viet Pham, Lin Tan, Xiangyu Zhang, and Michael W Godfrey. 2022. Docter: Documentation-guided fuzzing for testing deep learning api functions. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 176–188.
[70] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Hongxu Chen, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, Jianxiong Yin, and Simon See. 2018. Deephunter: Hunting deep neural network defects via coverage-guided fuzzing. *arXiv preprint arXiv:1809.01266* (2018).
[71] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. 2019. Diffchaser: Detecting disagreements for deep neural networks. International Joint Conferences on Artificial Intelligence Organization.
[72] Yuanmeng Yan, Rumei Li, Sirui Wang, Fuzheng Zhang, Wei Wu, and Weiran Xu. 2021. Consert: A contrastive framework for self-supervised sentence representation transfer. *arXiv preprint arXiv:2105.11741* (2021).

[73] Zhou Yang, Harshit Jain, Jieke Shi, Muhammad Hilmi Asyrofi, and David Lo. 2021. Biasheal: On-the-fly black-box healing of bias in sentiment analysis systems. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 644–648.

[74] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. 2020. White-box fairness testing through adversarial sampling. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 949–960.

Received 2023-09-28; accepted 2024-01-23