Leveraging ChatGPT to Predict Requirements Testability with Differential In-Context Learning

Mahima Dahiya, Rashminder Gill, Nan Niu
University of Cincinnati
Cincinnati, OH, USA
{dahiyama, randhars}@mail.uc.edu, nan.niu@uc.edu

Hemanth Gudaparthi
Governors State University
University Park, IL, USA
hgudaparthi@govst.edu

Zedong Peng
University of Montana
Missoula, MT, USA
zedong.peng@umt.edu

***Abstract*—Testability is a desired property of requirements, indicating how easy or difficult a requirements artifact supports its own testing. Prior work predicts natural language (NL) requirements' testability by training a decision tree (DT) via some readability and word measures. To explore better ways of predicting requirements testability, we examine in this paper large language models—ChatGPT in particular. Our experiments on a total of 1,181 requirements from six software systems show that ChatGPT's zero-shot learning performs worse than the DT. A main reason is due to the lack of context specific to the testability prediction task. However, applying ChatGPT's in-context learning (ICL) reveals a limitation of skewed examples caused by the imbalanced data. Thus, we propose a novel approach, called differential ICL, to address the challenges by exploiting the DT and show quantitatively the higher accuracy achieved by differential ICL.**

***Index Terms*—requirements testability, software testing, machine learning, large language models**

I. Introduction

Testability is the degree to which a software artifact supports testing in a given context [1]. A lower degree of testability results in increased test effort, and thus leads to less testing performed in practice. Beller *et al.* [2] showed that half of software developers in their study did not perform testing at all. The root of many software components not getting tested in practice lies in not considering testability early in the development process [3].

Surprisingly, little work has focused on requirements testability. In a survey of 208 papers, only 9 (or 4.3%) addressed issues related to requirements testability [4]. Among these 9 studies, Hayes *et al.* [5] pioneered the testability prediction of natural language (NL) requirements. They demonstrated the feasibility of building a supervised machine learning model—a decision tree (DT) in particular—by using a few attributes extracted from the NL requirements. These attributes include readability indices and word measures, which we detail in the next section. A 10-fold cross validation revealed that the DT achieved $\sim$70% accuracy in making binary predictions (testable or non-testable) of two datasets: Browser consisting of 23 total requirements and iTrust comprising an overall of 59 requirements.

A limitation of DT is its dependence on the specific dataset used in supervised learning. In contrast, foundation models like large language models (LLMs) use self-supervised learning over massive amounts of data to advance artificial intelligence (AI) in a dataset-agnostic direction. A prime example is ChatGPT which researchers have investigated to tackle various problems involving diverse data: program repair [6], fuzz testing [7], log parsing [8], just to name a few software engineering studies.

Recently, Gilardi *et al.* [9] reported that, for several text-annotation tasks including relevance and stance, zero-shot prompting of ChatGPT outperformed crowd workers in classification accuracy. This motivates us to prompt ChatGPT directly to predict whether or not a requirement is testable. Our experiments are conducted with not only the Browser and iTrust datasets [5], but also four more real-world software systems' NL requirements. Unfortunately, the results are not encouraging: Except for Browser (the smallest dataset), ChatGPT's zero-shot accuracy is worse than DT's accuracy.

One challenge of zero-shot classification is the lack of testability context. We thus leverage ChatGPT's in-context learning (ICL) capability [10], [11], aiming to embed a few examples in the prompt for better guiding the testability prediction. While the randomly chosen examples help improve the zero-shot accuracy on some requirements datasets, they lower the accuracy on the other datasets and still underperform the DT in general. We reveal an inherent reason being the data imbalance, i.e., testable requirements routinely outnumber non-testable ones, and thereby often creating an imbalanced context for ChatGPT's learning.

In this paper, we propose a novel approach called *differential ICL* to address the data imbalance problem, and apply it to the task of requirements testability prediction. Our core idea is to develop a new example to differentiate any given example. The differentiation traverses the DT to locate a decision node influencing the specific example, and then modifies that decision node's attribute value to create an oppositely labeled NL requirement. In this way, the result of differentiating a "testable" requirement is a "non-testable" requirement, and vice versa. Our approach thus uses a few *pairs* of examples and their corresponding differentiations, balancing ChatGPT's learning context. The experimental evaluations show that differential ICL consistently outperforms DT, zero-shot, and random ICL on all the six requirements datasets.

This paper makes two main contributions. First, we present a novel differential ICL approach that exploits ChatGPT to

2835-5776/24/$31.00 ©2024 IEEE
DOI 10.1109/IRI62200.2024.00044

TABLE I
SAMPLE REQUIREMENTS FROM THE BROWSER DATASET WHERE "ANSWER SET LABEL" DEFINES TESTABILITY: "1" MEANS THAT THE REQUIREMENT IS TESTABLE WHEREAS "0" MEANS THAT THE REQUIREMENT IS NON-TESTABLE

ID	NL Description	Answer Set Label
B1	Your browser must be implemented as a standalone Java SWT application on the Eclipse platform.	1
B21	Correct rendering of a successfully retrieved Simple HTML document shall mean the display of the contents of the document in the HTML display zone as specified in the Simple HTML specification section.	0

tackle the data imbalance challenge in many requirements engineering classification tasks. Second, by using 1,181 requirements of six software systems, we experimentally evaluate the accuracy of differential ICL, quantifying its effectiveness in predicting requirements testability. In what follows, we provide the background information and related work in Section II. We then use ChatGPT's zero-shot learning to predict requirements testability in Section III, present differential ICL in Section IV, and conclude the paper in Section V.

II. REQUIREMENTS TESTABILITY AND ITS PREDICTION

Organizations today invest more than one-third of their IT budgets in quality assurance and testing [12]. It is therefore pivotal to ensure the artifact produced in software engineering supports its own testing. However, not all software systems are easily testable. A fundamental reason is because testability is not considered early in the development process [3].

In requirements engineering, if no method can be devised to determine whether the software meets a particular requirement, then that requirement should be removed or revised [13]. As most requirements are written in NL [14], the work by Hayes *et al.* [5] has a wide scope of applicability and usefulness in practice. Thus, we regard Hayes *et al.*'s work as a state-of-the-art in testability prediction of NL requirements.

Hayes *et al.* [5] posit that a requirement is testable if a complete testing scenario used in functional testing can be found. They further hypothesize that existing NL quality measures like readability can be useful in predicting requirements testability. Low readability of a requirement implies it may not be testable, and seven readability measures are investigated in [5]: Average Grade Level (agl), Flesch Kincaid Reading Ease (fkre), Flesch Kincaid Grade Level (fkgl), Gunning Fog Score (gfs), SMOG Index (smog), Coleman Liau Index (cli), and Automated Readability Index (ari). Additionally, Hayes *et al.* [5] also incorporate three factors into creating predictive models for testability: the number of total words, complex words[1], and predicates/verbs in a requirement.

Using the Browser and iTrust datasets, Hayes *et al.* [5] predict whether a requirement is testable or non-testable by performing the correlation analysis, as well as the univariate and multivariate logistic regression analyses. The results indicate requirements testability prediction is unlikely to be linear with respect to the investigated factors, and also suggest some factors are less effective in predicting the 'testable'/'non-testable' label. Building on these results, Hayes *et al.* [5] construct a DT model with six attributes: fkre, fkgl, cli, now (number of total words), nocw (number of complex words), and pred (number of predicates/verbs).

[1]A complex word is any word with three or more syllables [5].

TABLE II
READABILITY AND NL ATTRIBUTES' VALUES OF THE SAMPLE BROWSER REQUIREMENTS LISTED IN TABLE I

ID	fkre	fkgl	cli	now	nocw	pred
B1	47.79	10.3	12.69	15	2	1
B21	30.54	17.0	13.94	32	6	3

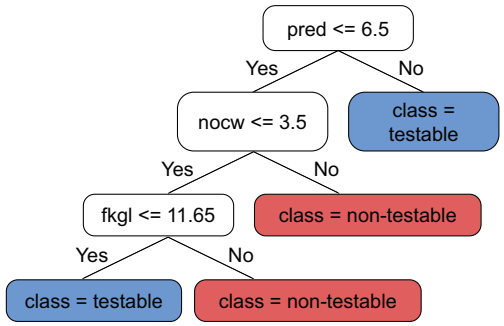

Fig. 1. DT trained on the Browser dataset.

We independently replicate the DT model described in [5], and illustrate this state-of-the-art method with a couple of requirements from the Browser dataset as shown in Table I. Since requirements testability is a subjective measure [5], we directly adopt the Browser and iTrust answer set labels from [5]. Each requirement is subject to the computations of the six attributes, as illustrated in Table II. The DT algorithm is then employed. During the training process, effective attributes and their distinguishing values are hierarchically organized into a binary tree, resulting in an inherently explainable representation of classification's decision making [19]. Fig. 1 shows a learned DT for the Browser dataset.

In summary, DT built by Hayes *et al.* [5] represents a state-of-the-art method in predicting the testability of NL requirements. Although the DT is intrinsically explainable, its construction is dataset and training-set dependent. To generalize testability prediction toward a dataset-agnostic and training-free direction, we next explore LLMs.

III. ZERO-SHOT LEARNING (ZSL) FOR REQUIREMENTS TESTABILITY PREDICTION

LLMs are deep neural networks that typically consist of 100 million to over 1 billion parameters. They are commonly obtained through self-supervised pre-training on massive corpora, e.g., Wikipedia, GitHub, arXiv, etc. In numerous instances, merely presenting the task description as a textual prompt

TABLE III
SUBJECT SYSTEMS AND THEIR DATASET CHARACTERISTICS: WHILE THE BROWSER AND ITRUST DATASETS ARE DIRECTLY ADOPTED FROM [5], THE DISCORD, FIREFOX, WEBEX, AND ZOOM DATASETS' REQUIREMENTS ARE COLLECTED FROM OCTOBER 2021 TO OCTOBER 2023

system [data source]	application domain	# of req.s	'testable' req.s in the answer set	'non-testable' req.s in the answer set	average # of words per req.
Browser [5]	web browser	23	17 (73.91%)	6 (26.09%)	60.5
iTrust [5]	electronic health records	59	53 (89.83%)	6 (10.17%)	63.5
Discord [15]	messaging & VoIP platform	117	86 (73.50%)	31 (26.50%)	30.5
Firefox [16]	web browser	179	128 (71.51%)	51 (28.49%)	26.5
Webex [17]	video conferencing	285	210 (73.68%)	75 (26.32%)	53.6
Zoom [18]	video conferencing	518	354 (68.34%)	164 (31.66%)	40.4

to pre-trained LLMs suffices for performing various natural language processing (NLP) tasks, eliminating the necessity for fine-turning [20], [21]. Such direct prompting of LLMs is referred to as zero-shot learning (ZSL).

In our task of requirements testability prediction, the default binary labels are 'testable' and 'non-testable' [5]. We follow the work of Gilardi *et al.* [9] who directly prompted OpenAI's ChatGPT for outputting a textual data point's class label. In the task of tweets' topic identification, Gilardi *et al.* [9] fed the tweets one by one to ChatGPT with the following prompt: "Here's the tweet I picked: [tweet's content], please label it as one of Section 230, Trump Ban, Complaint, Platform Policies, Twitter Support, and others". As a result, a tweet would be classified into one of the six classes.

Inspired by Gilardi *et al.*'s work, we feed the NL requirements one after another to ChatGPT by using the ZSL prompt: "Here's the requirement I picked: [NL description of the requirement]. Please label it as either testable or non-testable." This simple and direct prompt is appealing to us because crowd workers like those recruited via Amazon Mechanical Turk (MTurk) would be instructed in exactly the same way to perform text classification tasks [9]. Furthermore, Gilardi *et al.* [9] demonstrated that zero-shot ChatGPT classifications were more accurate than MTurk annotations, and therefore could potentially replace humans in text classification tasks. Will this also hold for requirements testability prediction? We report the experimental evaluations next.

A. Datasets and Their Answer Set Labels

We experiment six datasets which are listed in Table III. The Browser and iTrust datasets, along with their answer set labels, are directly adopted from [5]. Because data leakage is a major concern in leveraging an LLM to perform downstream tasks like requirements testability prediction, we must handle it carefully. Data leakage refers to the situation where the LLM has been trained on the experimental data so that it merely memorizes the results instead of predicting them. Following Gilardi *et al.*'s work [9], our research uses `gpt-3.5-turbo` which is trained with the data up to September 2021 [22]. Therefore, we mitigate the data leakage threat by collecting the requirements data released by Discord, Firefox, Webex, and Zoom from October 2021 to October 2023.

Hayes *et al.* [5] clearly pointed out that measuring requirements testability is a subjective and difficult task, and relied on researchers to manually assign the 'testable' or 'non-testable' label to the requirements in the Browser and iTrust datasets. We also engaged requirements engineering researchers in establishing the answer sets for the Discord, Firefox, Webex, and Zoom datasets. Specifically, two graduate students independently performed data labeling, guided with the very definition given by Hayes *et al.* [5]: "a requirement is testable if a complete testing scenario used in functional testing can be found". The two researchers were then joined by a third researcher in a collaborative meeting to discuss disagreements and reach consensus. Altogether, our manual labeling effort took approximately 65 human-hours, echoing the need to increase the efficiency of accurately predicting requirements testability [5].

B. Performance Measures

The DT's performance is assessed via a 10-fold cross validation [5]. The classification effectiveness is quantified by *accuracy*, calculated as the number of correct predictions divided by the total number of requirements in a dataset. As the predictions are about 'non-testable' and 'testable', accuracy considers both labels together.

To measure the quality of predictions separately on each label, we compute recall (R), precision (P), F_1-score (F_1), and F_2-score (F_2). Berry *et al.* [23] argue that, for automated tools supporting requirements engineering tasks in the context of large-scale software development, recall is more important than precision. Therefore, we include F_2-score which weighs recall twice as important as precision. Finally, the weighted R, P, F_1, and F_2 (represented respectively as wR, wP, wF_1, and wF_2) are calculated by using the distribution of the labeled classes in the dataset [24]. For instance, if the 'testable'-labeled and 'non-testable'-labeled requirements are distributed 70% and 30% in a dataset, the recall of 'testable'-predicted results is 0.86, and the recall of 'non-testable'-predicted results is 0.67, then the weighted recall of the total requirements is $\text{wR} = (70\% \cdot 0.86) + (30\% \cdot 0.67) = 0.80$.

C. Results and Analysis

To account for the randomness of DT's training-testing process, we execute the 10-fold cross validation ten times, and report (mean$\pm$standard deviation) in the second column of Table IV. Comparing DT's mean accuracy with ChatGPT's ZSL accuracy (cf. the third column of Table IV) shows that, except for the Browser dataset, DT outperforms ZSL. It is therefore unwise to abandon classic machine learning models

TABLE IV
REQUIREMENTS TESTABILITY PREDICTION ACCURACY BY DT (MEAN±STANDARD DEVIATION) AND BY CHATGPT'S ZSL, ALONG WITH CLASSIFICATION RESULTS BY CHATGPT'S ZSL FOR 'TESTABLE'-LABELED AND 'NON-TESTABLE'-LABELED REQUIREMENTS

subject	accuracy		ZSL on 'testable' req.s				ZSL on 'non-testable' req.s				ZSL on total req.s			
system	DT	ZSL	R	P	F_1	F_2	R	P	F_1	F_2	wR	wP	wF_1	wF_2
Browser	0.60±0.23	0.65	0.65	0.85	0.73	0.68	0.67	0.40	0.50	0.59	0.65	0.73	0.67	0.66
iTrust	0.78±0.11	0.63	0.62	0.94	0.75	0.67	0.67	0.17	0.27	0.42	0.63	0.86	0.70	0.64
Discord	0.65±0.09	0.47	0.37	0.80	0.51	0.42	0.74	0.30	0.43	0.57	0.47	0.67	0.49	0.46
Firefox	0.70±0.08	0.45	0.38	0.73	0.49	0.42	0.65	0.29	0.40	0.52	0.45	0.60	0.47	0.45
Webex	0.67±0.06	0.41	0.32	0.73	0.45	0.36	0.67	0.26	0.37	0.51	0.41	0.61	0.43	0.40
Zoom	0.60±0.03	0.49	0.43	0.71	0.53	0.47	0.62	0.33	0.43	0.53	0.49	0.59	0.50	0.49

supervised by labeled data. The performances achieved by these classic models should be used to critically assess LLM capabilities and quantitatively expose fallacies.

An interesting observation concerns an unintelligent predictor that classifies all the requirements as 'testable'. We call it PA1 (predicting all 1's). Due to the imbalanced data distributions (cf. Table III), PA1 would trivially match a majority of the requirements' labels, leading to accuracy = 0.74, 0.90, 0.74, 0.72, 0.74, and 0.68 for Browser, iTrust, Discord, Firefox, Webex, and Zoom respectively. Although these accuracies are better than what DT and ZSL achieve, PA1 offers no value in practice. Thus, it is crucial to analyze the performance metrics separately on the individual classes.

Such analyses are based on Table IV's columns of ZSL R, P, F_1, F_2, and their weighted counterparts. On the prediction of 'testable' requirements, ZSL's precision is higher than its recall for all the datasets. This implies ZSL's 'testable' predictions contain less noise but cover less ground truth as well. ZSL's 'non-testable' predictions, on the other hand, have a higher level of recall than precision. However, the precision levels are so low that more than half of the 'non-testable' requirements predicted by ChatGPT's ZSL are actually 'testable'. The noisy results, if presented to requirements engineers, would negatively influence their perceptions of ZSL's trustworthiness.

To gain insights into the ZSL mistakes, we manually inspect the misclassified cases. We make two observations about the under-par ZSL performances: (1) not having any context of the specific classification task, and (2) not understanding the concept conveyed in single word like 'testable'. We propose a novel approach to overcoming these problems in Section IV.

D. Threats to Validity

The construct validity of our ZSL study can be affected by our direct prompting of the LLM. Embedding-based ZSL, on the other hand, makes the final classification decision on the basis of the Cosine similarity of requirements and class labels [24]. Our manual labeling might introduce errors, though considerable effort (∼65 human-hours) had been expended to obtain high-quality answer sets.

A threat to internal validity is our DT implementation which does not handle the skewed class proportions. The main reason is that the steps undertaken by Hayes *et al.* [5]—boosting, bagging, and blending—are shown to be ineffective. Furthermore, ZSL's performance does not depend on the data size of each class, because no actual learning is performed on the labeled data [24].

Threats to the external validity include our investigations of six requirements datasets. In addition, our ZSL uses only gpt-3.5-turbo, though the method can be replaced as long as the LLM or the relevant API is accessible.

IV. DIFFERENTIAL IN-CONTEXT LEARNING (ICL)

One of our major lessons learned from Section III is to provide the proper context for the LLM to perform the requirements testability prediction task. To that aim, we take advantage of LLM's in-context learning (ICL). Since the advent of GPT-3, researchers have noted that LLMs possess the capability of learning in the prompt context to perform inference. This few-shot prompting is referred to as ICL [10], [11]. Another important lesson is not to overly rely on the simple class labels in defining the task context. Therefore, we use '1' and '0' to frame the ICL demonstrations, as shown in Fig. 2. Clearly, a practical question of ICL is how many demonstrations should be provided. While the number needs to be *a few*, Gao *et al.* [25] empirically suggested the optimal demonstration quantity to be four in software engineering tasks. We thus adopt four ICL demonstrations in our work, as illustrated in Fig. 2. Our future work plans to investigate the quantity of learning examples in requirements engineering.

To build a desired context, we propose to pair a learning example with its oppositely-labeled variant. Such pairs not only balance the task context in terms of the labeled examples'

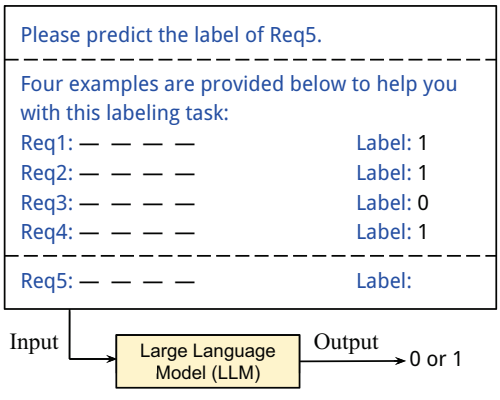

Fig. 2. Using four examples (Req1–Req4) to demonstrate the task context for the LLM to predict whether the target requirement (Req5) is testable.

quantities, but also enable the differentiation between the two examples within a pair. We call this approach *differential ICL* (dICL). In the scheme outlined in Fig. 2, dICL no longer randomly chooses four demonstrations, but selects two randomly, e.g., Req1 with label=1 and Req3 with label=0. Then, dICL forms two pairs by creating Req1' with label=0 and Req3' with label=1. Finally, dICL uses these four requirements—{Req1 (label=1), Req1' (label=0), Req3 (label=0), Req3' (label=1)}—as the examples to demonstrate the labeling task to be completed by the LLM on Req5. It is important to note that, though Fig. 2 uses {Req1, Req2, Req3, Req4} to illustrate ICL with randomly chosen examples, our method of dICL uses two randomly selected examples (e.g., Req1 and Req3) and their differentiations (i.e., Req1' and Req3') to form the balanced learning context: {Req1, Req1', Req3, Req3'}.

One automatic way of devising the oppositely-labeled variant (req') for any given requirement (req) is to manipulate the traversal of req along the DT resulted from testability classifications [5]. Algorithm 1 depicts the procedure. Lines 3–4 of Algorithm 1 show that, in case the DT's testability classification of req is correct, the most specific decision node right before reaching the leaf will be operated to create req'. If DT's classification is incorrect, then traversal.leaf.parent itself will not be a viable choice. Line 6 of Algorithm 1 then determines what to operate based on two conditions of a decision node: (i) having a leaf with the same label as req', and (ii) having the shortest distance to traversal.leaf.parent in the DT. While condition (i) ensures the desired label is achievable, the intent of condition (ii) is to preserve the operational node's specificity as much as possible.

Once opn_node is determined, line 8 of Algorithm 1 applies it to req in order to create req'. We illustrate this step via the Browser dataset's B1 requirement (cf. Tables I and II) and DT (cf. Fig. 1). B1 traverses all the left branches of Fig. 1's DT, landing on a '1'-labeled leaf node. Because this leaf node's label matches the answer_set_label of B1, lines 3–4 of Algorithm 1 are executed. The leaf node's parent, namely "fkgl <= 11.65", is designated as the operational node for B1. To create B1' is to change B1 by switching the decision made on "fkgl <= 11.65". Since B1 traverses "Yes" here, we want B1' to go to "No". Therefore, we search all the Browser dataset's requirements that satisfy "fkgl > 11.65", and then choose the most similar one[2] to B1 from the search results. In this case, B15 with fkgl=15.4 is selected. Finally, B1' is created by replacing the first five terms in B1 with those in B15. Note that the size of five terms is not a parameter but a property of the English language [26], [27]. Replacing the *first* five terms is due to our preserving the beginning of each requirement's NL description in case the token limit of gpt-3.5-turbo needs to be dealt with. Corresponding to B1 shown in Table I, B1' is: "Help --> About. When this as a standalone Java SWT application on the Eclipse platform."

Algorithm 1: Creating Differential Example

```
Input: req  // an ICL example
Input: DT  // the testability decision tree trained on
       the requirements dataset
Output: req' // differential example corresponding to
        req such that the label of req' is opposite to
        the label of req
1 opn_node ← DT.root // set DT's root node to be the
    default operational node (opn_node)
2 traversal ← traverse req over DT
3 if traversal.leaf.label == answer_set_label(req) then
4 |   opn_node ← traversal.leaf.parent
5 else
6 |   opn_node ← decision_node that leads to the label
  |     of req' and that is closest to traversal.leaf.parent
7 end
8 return opn_node(req)
```

The novelty of dICL lies in the full automation of creating req' for any given req. Besides, the creation is efficient due to the linear search of the binary DT to locate the operational node. To evaluate the ICL performances, we test the random way of choosing the four demonstrations and call this method rICL. For each requirement, we run rICL 30 times and deem rICL's prediction result to be '0' (respectively '1'), if more than half of the runs output '0' (respectively '1'). We use the same majority rule for dICL, which is also run 30 times on each requirement.

The performance averaged over 30 runs is summarized in Table V. Compared to rICL, dICL performs better in all the measures on all the datasets. Based on the experimental results, we conclude that rICL and ZSL have the lowest prediction accuracies, DT achieves better accuracy levels, and dICL predicts requirements testability most accurately.

Threats to validity of our ICL experiments include running either variant of rICL and dICL 30 times. The law of large numbers states that if the sample size is sufficiently large (30 to 50 samples), then the central limit theorem applies [28]. Thus we expect the 30 runs to indicate statistical significance. Although the significance is supported in our six experimental datasets, the results may not generalize to requirements from other systems thereby affecting the external validity.

V. Conclusions

Testing remains a common method for uncovering faults and assessing dependability of software-intensive systems [5], [29]. A hallmark of high-quality requirements is that they shall be testable [13]. In this paper, we have experimentally investigated ChatGPT's ZSL and ICL in predicting NL requirements' testability. The key take-away is that a traditional AI method, notably DT [5], performs better than ZSL and ICL instrumented by randomly chosen examples.

We present a novel approach that leverages DT to develop differential learning examples, and show that differential ICL outperforms DT, ZSL, and random ICL in all the six requirements datasets under investigation. Our future work

[2]Informed by Alhoshan *et al.*'s work [24], we use Cosine similarity.

TABLE V
PERFORMANCES OF TWO VARIANTS OF IN-CONTEXT LEARNING (ICL) FOR PREDICTING REQUIREMENTS TESTABILITY WHERE THE DEMONSTRATIONS ARE SELECTED IN A RANDOM (rICL) OR DIFFERENTIAL (dICL) MANNER

subject system and method		accu -racy	ICL on 'testable' req.s				ICL on 'non-testable' req.s				ICL on total req.s			
			R	P	F_1	F_2	R	P	F_1	F_2	wR	wP	wF_1	wF_2
Browser	rICL	0.48	0.47	0.67	0.55	0.50	0.50	0.50	0.50	0.50	0.48	0.62	0.54	0.50
	dICL	0.78	0.76	0.76	0.76	0.76	0.83	0.56	0.67	0.76	0.78	0.71	0.74	0.76
iTrust	rICL	0.59	0.58	0.94	0.72	0.63	0.67	0.15	0.25	0.40	0.59	0.86	0.67	0.61
	dICL	0.79	0.79	0.98	0.88	0.82	0.83	0.31	0.45	0.63	0.80	0.91	0.83	0.80
Discord	rICL	0.56	0.47	0.87	0.61	0.51	0.81	0.35	0.49	0.64	0.56	0.73	0.58	0.55
	dICL	0.70	0.63	0.95	0.76	0.67	0.90	0.47	0.62	0.76	0.70	0.82	0.72	0.70
Firefox	rICL	0.57	0.48	0.86	0.61	0.52	0.80	0.38	0.52	0.66	0.57	0.72	0.59	0.56
	dICL	0.75	0.70	0.94	0.80	0.74	0.88	0.54	0.67	0.78	0.75	0.82	0.77	0.75
Webex	rICL	0.56	0.45	0.90	0.60	0.50	0.87	0.36	0.51	0.68	0.56	0.76	0.57	0.54
	dICL	0.70	0.62	0.96	0.75	0.67	0.92	0.46	0.62	0.77	0.70	0.83	0.72	0.69
Zoom	rICL	0.54	0.48	0.75	0.58	0.52	0.66	0.37	0.48	0.57	0.54	0.63	0.55	0.53
	dICL	0.65	0.58	0.86	0.69	0.62	0.79	0.47	0.59	0.70	0.65	0.73	0.66	0.65

includes experimenting with more requirements datasets [30], performing replication studies [31], [32], and expanding from assessing an individual requirement's testability to feature interactions [33], [34].

ACKNOWLEDGMENT

We thank Jane Hayes and her colleagues for the foundational work done in applying machine learning to predict requirements testability, and for sharing their labeled Browser & iTrust datasets with us.

REFERENCES

[1] J. M. Voas and K. W. Miller, "Software testability: The new verification," *IEEE Software*, vol. 12, no. 3, pp. 17–28, May 1995.
[2] M. Beller *et al.*, "Developer testing in the IDE: Patterns, beliefs, and behavior," *IEEE Transactions on Software Engineering*, vol. 45, no. 3, pp. 261–284, March 2019.
[3] M. Ghafari *et al.*, "Testability first!" in *ESEM*, Porto de Galinhas, Recife, Brazil, September 2019.
[4] V. Garousi *et al.*, "A survey on software testability," *Information & Software Technology*, vol. 108, pp. 35–64, April 2019.
[5] J. Hayes *et al.*, "Measuring requirement quality to predict testability," in *AIRE*, Ottawa, Canada, August 2015, pp. 1–8.
[6] M. Jin, *et al.*, "InferFix: End-to-end program repair with LLMs," in *ESEC/FSE*, San Francisco, CA, USA, December 2023, pp. 1646–1656.
[7] C. S. Xia *et al.*, "Universal fuzzing via large language models," *CoRR*, August 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.04748
[8] V.-H. Le and H. Zhang, "Log parsing: How far can ChatGPT go?" in *ASE*, Kirchberg, Luxembourg, September 2023, pp. 1699–1704.
[9] F. Gilardi *et al.*, "ChatGPT outperforms crowd-workers for text-annotation tasks," *Proceedings of the National Academy of Sciences*, vol. 120, no. 30, 2023.
[10] T. B. Brown, *et al.*, "Language models are few-shot learners," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, Held Virtually, December 2020.
[11] Q. Dong *et al.*, "A survey on in-context learning," *CoRR*, June 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2301.00234
[12] C. Ebert *et al.*, "Testing software systems," *IEEE Software*, vol. 39, no. 4, pp. 8–17, July/August 2022.
[13] IEEE STD 830 *(Reaffirmed: September 12, 2009)*, "IEEE Recommended Practice for Software Requirements Specifications," Last accessed: June 7, 2024. [Online]. Available: https://doi.org/10.1109/IEEESTD.1998.88286
[14] M. Kassab *et al.*, "State of practice in requirements engineering: Contemporary data," *Innovations in Systems and Software Engineering*, vol. 10, no. 4, pp. 235–241, December 2014.
[15] Discord, "Change Log," https://discord.com/developers/docs/change-log, 2023, Last accessed: June 7, 2024.
[16] Mozilla Foundation, "Mozilla Firefox Release Notes," https://www.mozilla.org/en-US/firefox/releases/, 2023, Last accessed: June 7, 2024.
[17] Cisco, "Webex App: What's New," https://help.webex.com/en-us/article/8dmbcr/Webex-App-—-What's-New, 2023, Last accessed: June 7, 2024.
[18] Zoom, "Release Notes for Windows," https://support.zoom.us/hc/en-us/articles/201361953-Release-notes-for-Windows, 2023, Last accessed: June 7, 2024.
[19] N. Maltbie, *et al.*, "XAI tools in the public sector: A case study on predicting combined sewer overflows," in *ESEC/FSE*, Athens, Greece, August 2021, pp. 1032–1044.
[20] P. Liu *et al.*, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 195:1–195:35, September 2023.
[21] J. Zhang, *et al.*, "Empirical evaluation of ChatGPT on requirements information retrieval under zero-shot setting," in *ICNGN*, Hangzhou, China, November 2023.
[22] OpenAI, "GPT models," 2023, Last accessed: June 7, 2024. [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5
[23] D. M. Berry *et al.*, "Panel: Context-dependent evaluation of tools for NL RE tasks: Recall vs. precision, and beyond," in *RE*, Lisbon, Portugal, September 2017, pp. 570–573.
[24] W. Alhoshan *et al.*, "Zero-shot learning for requirements classification: An exploratory study," *Information & Software Technology*, vol. 159, pp. 107 202:1–107 202:15, July 2023.
[25] S. Gao *et al.*, "What makes good in-context demonstrations for code intelligence tasks with LLMs?" in *ASE*, Kirchberg, Luxembourg, September 2023, pp. 761–773.
[26] Y. S. Maarek *et al.*, "An information retrieval approach for automatically constructing software libraries," *IEEE Transactions on Software Engineering*, vol. 17, no. 8, pp. 800–813, August 1991.
[27] N. Niu and S. Easterbrook, "Extracting and modeling product line functional requirements," in *RE*, Barcelona, Spain, September 2008, pp. 155–164.
[28] R. M. Sirkin, *Statistics for the Social Sciences*. SAGE Publications, 2005.
[29] Z. Peng *et al.*, "Testing software's changing features with environment-driven abstraction identification," *Requirements Engineering*, vol. 27, no. 4, pp. 405–427, December 2022.
[30] M. Dahiya *et al.*, "Tracing feature tests to textual requirements," in *IRI*, San Jose, CA, USA, August 2024.
[31] N. Niu *et al.*, "Advancing repeated research in requirements engineering: a theoretical replication of viewpoint merging," in *RE*, Beijing, China, September 2016, pp. 186–195.
[32] C. Khatwani *et al.*, "Advancing viewpoint merging in requirements engineering: A theoretical replication and explanatory study," *Requirements Engineering*, vol. 22, no. 3, pp. 317–338, September 2017.
[33] W. Wang *et al.*, "Detecting software security vulnerabilities via requirements dependency analysis," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1665–1675, May 2022.
[34] Z. Peng *et al.*, "Resource-centric goal model slicing for detecting feature interactions," in *IRI*, Bellevue, WA, USA, August 2023, pp. 58–63.