



VORI: A framework for testing voice user interface interactability

Abrar S. Alrumayh*, Chiu C. Tan

Department of Computer and Information Sciences, Temple University, Philadelphia, USA

ARTICLE INFO

Keywords:

Voice user interface
Smart speaker
Usability
Home voice assistant

ABSTRACT

The ability of Voice User Interface (VUI) to understand how users will express their commands naturally and intuitively is an essential component of user experience, especially when the user is interacting with the VUI for the first time. Designing an automated method for testing the usability of VUI is a challenge for two reasons. First, there are many different ways for a user to express the same intention, e.g. “play some music”, “put some music on”, etc., that is difficult to determine in advance. Second, many VUI apps today typically rely on the platform service provider (e.g. Amazon, Google, etc.) to perform many of the speech recognition and natural language processing tasks, and these services are provided as a blackbox. Consequently, it is difficult for the app developer to obtain information about errors and user feedback. In this paper, we propose a framework, VORI, to systematically evaluate the interactability of VUI, as well as a new metric for quantifying the interactability of a VUI. We use VORI to analyze 127 applications on Alexa by sending over 82,931 commands. Our analysis results highlight that 41.7% of apps only accept strict input that has to exactly match the developer’s predefined sample commands with an interactability score of 20% or less. This suggests developers should consider a better interactability strategy in the design of VUIs, and more research is needed to further explore the design space to improve the interactability.

1. Introduction

Voice user interfaces (VUIs) are increasingly being found on platforms such as automobiles, head-mounted devices, and home smart speakers. Unlike earlier VUIs found on platforms like laptops, where the VUI complements other interfaces (e.g. keyboard) to provide hands-free interaction with the device, these emerging platforms have VUIs as the primary means of interacting with the device. This means the usability of VUIs is more important than before. A representative platform of this new voice-only paradigm is smart speakers like Amazon Alexa [1] and Google Home [2]. Approximately 60 million households now own a smart speaker [3], and more than 100 million Alexa devices have been sold [4]. Given their popularity, this paper will consider VUIs from the perspective of Alexa smart speakers, though our techniques can be extended to other VUIs as well.

Smart speakers support an ecosystem similar to smartphones, where the platform developer (e.g. Amazon, Google, Microsoft) provides an extensive API to allow 3rd party developers to develop smart speaker apps on their platform. Users can enable these apps from an app store, just like smartphone apps, to provide various services, including controlling other smart-home devices (e.g. TV, lighting), aid services (e.g. creating calendar reminders, setting alarm), and even performing virtual services (e.g. bank transfer, ordering takeout, hail a cab, etc.). Amazon market

now includes more than 100,000 of such apps, and the Google market has over 4,000 apps [5]. This diverse set of 3rd party developers means that there is no consistent “interface” that users can expect since each 3rd party developer could implement their own voice commands differently from each other. From the app developer’s point of view, this lack of a standard interface makes designing smart speaker apps challenging, since there are many different ways of how someone will express the same intention, based on their demographics [6], age [7], accent [8–10], and so on. This makes it difficult for users to remember what commands a particular app can process [11]. The architecture of current smart speaker apps further complicates the testing process since the app is hosted on the smart speaker platform as a black-box as shown in Fig. 1. If there is some sort of error the voice interface typically returns a default response (e.g. “I’m sorry, I didn’t quite get that”) and quits. This error could be because of background noise interference, the way the command was phrased, the user’s accent, network failure, and so on. This makes it difficult for the developer to accurately determine the root causes.

To help improve the usability testing for VUIs, we designed a testing framework, called VORI (Testing VOice useR Interface Interactability). VORI will automatically and systematically evaluate the interactability score to determine how well a VUI has been implemented to accept different kinds of user interaction. We envision that app developers can

* Corresponding author.

E-mail addresses: abrar.alrumayh@temple.edu (A.S. Alrumayh), cctan@temple.edu (C.C. Tan).

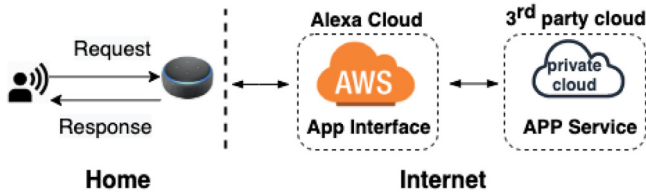


Fig. 1. The architecture of Echo System where the app is hosted as a black-box.

use VORI to measure how well their particular VUI stacks up against similar kinds of apps to improve their design. Our main contributions are as follows.

- Unlike existing VUI testing tools which are limited to testing certain APIs (e.g. apps that involve the alarm API cannot be tested), VORI is able to be used to test *any* smart speaker apps.
- We propose the idea of an *interactability score* to quantify how well a VUI can accept potentially different ways a user may express their commands. This score provides a convenient way of comparing different apps.
- We leverage the intelligence of all the developers of *similar* apps to arrive at a list of possible utterances representing the different ways users could say things. We use natural language processing (NLP) techniques to the collected commands to derive a larger set of commands for testing.
- We applied VORI to analyze 127 applications on Alexa by sending over 82,931 commands to test, and analyze 89,635 responses. We find that about 40% of apps only accept commands that exactly match the developer's predefined sample commands, which suggests the VUI will result in a poor user experience.

Note that while our current prototype focuses on the Alexa platform, our proposed techniques can be expanded to other platforms as well. The rest of the paper is organized as follows: The related work is discussed in Section 2. Then, we introduce the interactability metric in Section 3. Section 4 presents our testing framework design, followed by evaluation Section 5. Finally, Section 6 presents our additional discussion and conclusion.

2. Related work

With the growth of smart speakers, a large group of studies focuses on understanding how users interact with VUIs, and what challenges they face with the aim of improving user satisfaction and learnability. Users face challenges due to underlying software components ranging from natural language processing, error control to system responsiveness [11]. Purington et al. [12] examine VUI sentiment in Amazon customer reviews of Alexa's Echo in order to identify characteristics that lead to higher user satisfaction. According to their research, users prefer the personification of VUIs and this subjectivity can improve the user experience. Kirschthaler et al. [13] claimed that users struggle to interact with VUI due to, difficulty to discover and learn commands or forgetting commands. A consistent challenge is determining what people can do with the interface through voice command. Users had difficulty issuing long commands, or commands had to be issued multiple times in order to complete the task [8]. This problem is exacerbated by allowing third-party developed extensions to be made available, often with varying and inconsistent commands. As a result of this discoverability issue, users may fail to form an appropriate mental model about the VUI, which results in a poor user experience [13]. Some may argue that rather than requiring developers to come up with plenty of variation, users should memorize specific phrases. However, for good usability and a better user experience would be for users to choose their own phrases [14]. The existing literature indicates that there are still usability issues; to address this, we developed VORI, an automated tool for understanding the usability obstacles in interacting with VUIs.

Discoverability which is defined by how easy it is for users to find and execute features of VUIs has received attention in the literature. To improve the learnability and discoverability of VUIs, existing research has focused on new interaction techniques that allowing users to ask a range of possible questions [15], and adaptive and contextualized learning [16]. Recently [13] has revealed that strategies for increasing discoverability, such as making commands available via phrases like "what can I say?" or either explicitly prompting users, significantly improve usability scores. In comparison, our work also strives to understand the usability of VUIs, but our approach is different in what we are measure. Interactability measures how well the VUI will understand what a user will say naturally without the user having to learn anything at all. The rationale for measuring interactability is that users may have different ways of communicating with an app, and a good audio interface should be able to recognize and implement these different ways.

Work by Lentzsch et al. [17], Alrumayh et al. [18] explore the security and privacy risks on third-party applications that are built on top of Alexa and identifying the limitations in the current app vetting process. Another work [19] measure the effectiveness of privacy policies provided by the app developers. Existing works that adopted Natural language processing (NLP) techniques to measure the trustworthiness of app certifications [20]. Recently there is interest on building automatic testing tool, work by Guo et al. [21] explore the behaviors of voice apps to detect privacy violations in apps, and they found that many apps request users' private information. While [22] proposed testing tool to investigate the health-related applications on Alexa and how well they comply with existing privacy and safety policies. These initial efforts all rely on the simulator tool developed by Amazon for their Alexa platform that allows developers to communicate with an app using texts and observe its outputs also in text form. However, this text-based simulator cannot be used to test audio aspects such as accents and pronouncements, and is restricted to certain APIs. Apps that involve some type of alarms, for instance, cannot be tested using the simulator. Comparing to this work, we focused on usability metric by quantifying how well well a VUI is able to predict what commands an end user will use to interact with the interface.

3. Interactability as a usability metric

The idea behind **interactability** is to measure how well a VUI is able to match what commands someone will use to interact with the interface. There are many different ways of how someone will express the same intention, based on their demographics [6], age [7], accent [9,10], and so on. Take for instance, a VUI for an alarm app. A user may set the alarm by saying, "set the alarm for 7 in the morning", "wake me up at 7 am", "wake me up in 8 hours", "tell the alarm I want to wake up at 7 in the morning", "wake me up to [song name] by [artist] at 7 am", etc.. Since the VUI developer can only program the correct responses based on the developer's prediction of what users' might say to interact with the app, the more accurate the developer's predictions are, the better the end-user experience is going to be.

The interactability score is a means of quantifying how well this prediction is made. A VUI with a high interactability score will be able to respond to a more diverse set of commands, than a VUI with a low interactability score. More formally, we define the interactability score (*interact_score*) as

$$interact_score = \frac{\Theta_{utterances}}{\Theta_{input}} \cdot 100 \quad (1)$$

Where Θ_{input} is the set of what users might say to the VUI, and $\Theta_{utterances}$ is the set that represents what the VUI developer predicts users might say when using the VUI.

Interactability and other metrics. We believe the idea of interactability can fill in a gap in existing VUI usability evaluation. Traditional metrics from NLP, like word error rate (WER), match error rate (MER), and word information lost (WIL) [23], are helpful in determin-



Fig. 2. An example of modifying voice commands.

ing how accurate the VUI can translate human speech into text. While the accuracy is an essential component in determining the usability of a VUI, it is insufficient, since the app design may not have anticipated a particular way of expressing a command, and hence, the accurate translation will still result in an error. More recent work on VUI has proposed the idea of *discoverability* [13], a measure of how easy it is for users to find and execute features of VUIs. Discoverability is a useful metric because there are no conventions on how users will typically express standard commands such as searching for information, saving a file, and so on, nor is there a standard vocabulary that users are expected to use for these common tasks [24]. The easier it is for a user to learn what commands the VUI will understand, the better the user experience.

Interactability is different from discoverability in that we are attempting to measure how well the VUI will understand what a user will say naturally without the user having to learn anything at all. This will increasingly be important since platforms like Amazon Alexa support “apps” or “skills” from third-party developers. Looking at the trend from smartphones, we can expect that users may install dozens, if not hundreds, of apps that rely on VUI. Users may not want to learn how to interact with each app individually. An app with a good interactability score that *just works* will provide a better user experience.

How developers manage interactability? The general guidelines when designing a voice app will recommend developers to build a set of likely spoken phrases, which should include as many representative phrases as possible [25]. The app’s usability directly depends on how well the sample utterances and custom slot values represent real-world language use. However, there is currently no mechanism for developers to determine whether their set of representative phrases is adequate.

Furthermore, the current architecture of VUIs in general (Fig. 1), makes it difficult for developers to collect user errors to improve their products. On the Alexa platform, for example, when a user sends a request, Alexa will try to match with one of the predefined lists of utterances, if the developer did not implement a particular vocabulary term or expression, or if the user clearly asks for a value not spelled out, Alexa typically returns a default error response (e.g. “I’m sorry, I didn’t quite get that”) before quitting. To handle these unexpected utterances “out-of-domain requests”, or when a customer says something that doesn’t map to any intents, developers are allowed to implement AMAZON.FallbackIntent built-in library that lets them respond gracefully to an out-of-domain request. They can provide additional instructions or sample utterances on what the app does and reorient customers. In this case, developers will be informed there is an out-of-domain request but will never know the “exact utterance the user says” that causes the error.

4. VORI framework for testing voice user interface

While the computation of the interactability score (Eq. (1)) appears straightforward, designing a framework to automate the testing process has its own challenges. The usefulness of the score depends on how well Θ_{input} reflects what users will say in the real world. A straightforward approach of using a small-scale user study is unlikely to capture the

diverse ways of communicating the same piece of information, while a larger-scale user study, will be expensive to conduct for every app that utilizes a VUI.

Our VORI framework addresses this problem by leveraging the collective intelligence of multiple developers. The intuition is that since each individual app developer has put in the effort to predict how a user might interact with his specific app, our approach is to compute Θ_{input} by aggregating the sample commands from *similar* apps made by multiple different developers. For example, to test an app that reminds the user to workout, we will collect the sample commands from other reminder apps (e.g. remember to buy milk, do homework, etc.), and then process these commands to compute this list of possible commands for the workout reminder app.

4.1. Design of VORI framework

The VORI design can be broken down into three components: *command collection*, *command modification*, and *response analysis*. We cannot directly use voice commands from other apps, since these commands are generally tailored for specific apps. For example, an exercise reminder app will not understand the commands for a medication reminder app, even though both are reminder apps.

Collecting voice commands Amazon Alexa allows developers to provide command samples on their apps’ introduction pages in the store to help users understand how to use the app, hoping to make the app easy to use. We automatically crawl the introduction page and extract sample command inputs to initialize the interaction. The standard position of the sample command can be located by using “a2s-utterance-box-inner” in the source code of the web-page. Also, we found that some developers include additional sample commands in the app description which usually appear in double-quotes or in the form of lists. To extract these commands from the app description, we followed the process used by Shezan et al. [22]. We used a web crawler that collects sets of valid utterances for VUIs from similar apps and subsequently generates the wealth of possible commands across various apps. We consider all apps under the same category in the app store are similar. For example, on the Alexa app store [26] under the “Productivity” there are “Alarms & Clocks”, “Calendars & Reminders”, and “To-Do Lists & Notes” categories, therefore we treat all apps in each individual category as similar apps. **Modifying voice commands** After collecting the commands, we do additional processing to increase the number of testing cases and allow cross app evaluation, i.e. take command for creating a reminder to take a medicine and test in another app that creates reminder recycling week, since both apps are supposed to accomplish similar tasks. Fig. 2 (a), we

modify the command to take medicine “Alexa, ask Remind me I want to remember to take my pill next Tuesday 7 pm” to “Alexa, ask Trash Day I want to remember the recycling week for Green bins next Tuesday 7 pm”. However, in some cases replacing the event “take my pill” with the type of events supported by the target app “recycling week..” is more challenging. As some apps enforce specific keywords or structures for

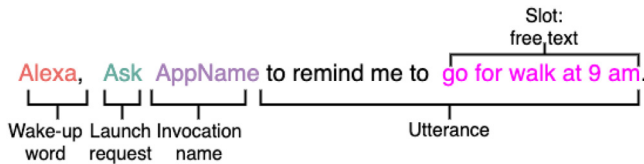


Fig. 3. Parsing user commands into invocation name, utterance, and slot.

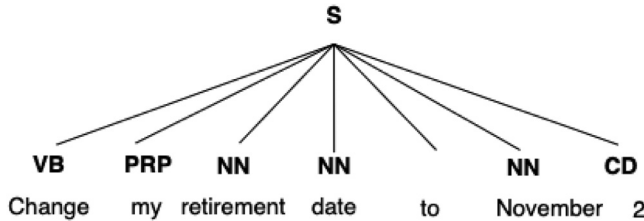


Fig. 4. An examples of Part-of-Speech tagging parsing the command "change my retirement date to November 2".

commands. As a consequence, the structure of input commands has to be carefully designed. Fig. 2 (b) shows an example of an app that strict users to say the keyword "Record Memo".

Fig. 3 illustrates the components of the voice command to create a reminder. Each command has four main parts: the wake-up word, launch request, invocation name, and utterance. The wake word is the voice service's name (Alexa) that puts the Alexa into the listening mode to take the users' requests. The invocation name is the app name. Utterances are phrases the users will use to make a request which identifies what the user wants the app to perform. Utterances may contain a slot that represents free text entered by the user to give the app more information about that request. We take each of the documented commands from all apps and generate corresponding commands for each app. The first two parts of the command, wake-up word, and launch request, will not change. The third part, invocation name, simply replaces it with the target app's name. Our main focus is the fourth part, the utterance.

In order to automatically generate the corresponding utterances, we used a Natural Language Processing (NLP) approach to analyze the semantic meaning and understand each word of the utterance. Therefore, a suite of NLP processes is applied to the crawled commands to derive a larger set of commands by keeping grammar and only changing keywords. Fig. 7 depicts the process of commands generator. Specifically, we apply *Part-of-Speech tagging* (POS), followed by *Shallow Chunking*, then *Entity Recognition*. POS is a process of deciding the type of every word in the text (nouns, verbs, adjectives, etc.) based on its definition (Fig. 4). While shallow chunking is a process of analyzing the structure of a sentence and linking words to syntactically related groups that makeup phrases (Fig. 5). Finally, entity recognition is the information extraction process to obtain a list of named entities such as the name of a person, date or time, number, place, organization or product, etc. This is likely the same technique used by Amazon on recognizing patterns and meaning within users' commands [27]. In order to identify the keyword which needs to be modified, we focus on the "NP" tag embedded inside the "VP" tag. Thus, to identify the task or record that needs to be set, adjust, look-up, etc through using the NLP, we first locate the "NP" tag and then replace those events with the type of events supported by the target app, e.g. water the planet, as a keyword to replace. **Analyzing VUI responses.** After the command is sent to an app, it will feedback a response. VORI should understand these responses in order to evaluate the interactability. To understand diverse responses, we should classify responses into different types, and further analyze usability according to their types. Based on the interactability score (*interact_score*) of this paper, we classify app responses into three types: failure, recognized, and help. If an app fails to recognize the input command, the app will provide an error

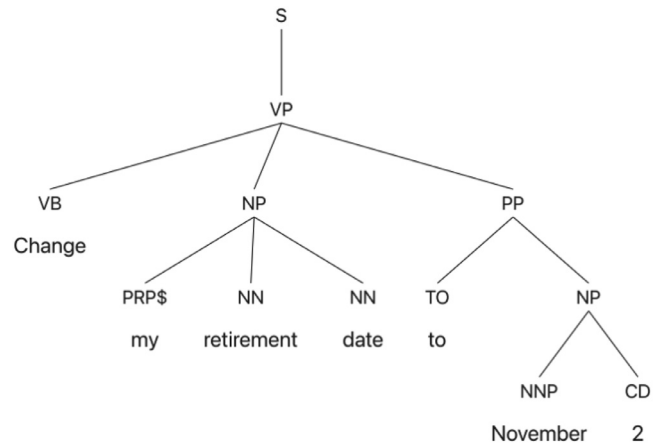


Fig. 5. An examples of Shallow Chunking parsing the command "change my retirement date to November 2".

Table 1
Example rules for response understanding.

Rule	Situations	Response Type
R1	"My apologies, please repeat your command", "sorry, I don't understand", "I am having trouble understanding", "I could not find what you ask for", "I don't get that", "I am unable to find that"	Failure
R2	statement, wh-questions, "okay I'll remind you..", "created successfully", "Your [reminder, task, record] created successfully", "when should I remind you", "What the reminder for"	Recognized
R3	"You can say..", "say", "ask", "just ask..", "Here are some things you can say", "Please say"	Help

message such as "sorry, I don't understand that". If the app *successfully recognizes* the input command, the app's response will be in the form of a statement such as "your reminder created successfully", or ask further questions to proceed with the user's request such as "when should I remind you". If the app is not programmed to recognize the input command, the app will respond with *help* by giving informative instructions that guide users to say the correct command such as "You can say..", or "just ask for". We summarize the rules as shown in Table 1. These three types help in quantifying how well the prediction is done. We defined the $\Theta_{utterances}$ as the number of *recognized* and *help*:

$$\Theta_{utterances} = \sum R + \sum H \quad (2)$$

Where R is the recognized responses, and H is the help responses.

We develop a rule-based approach because the app's responses usually have clear patterns, such as using the words "ask" or "say" as instruction responses for help requests. The main reason is the samples given by Amazon for developers as design guidelines to build apps [28]. For example, in case of failure or unaccepted command, the app typically returns a default response, "I'm sorry, I didn't quite get that" [8]. In order to create a repository for those responses, we interact with 60 randomly selected apps using the commands database and collect the responses. Then we manually analyze, label, and build a repository of responses related to the three testing metrics.

Then we define matching rules based on the three rules in order to decide which pattern is used in the response. In particular, we use Python library for rule-based matcher in NLP. The first thing we define the patterns that we want to match (Table 1). Next, we add the patterns to the Matcher tool, then we apply the Matcher tool to the responses output file generated by the interaction tool. Finally, the matcher looks for patterns that we have encoded as a knowledge base. It processes word by word by first lemmatization each word to its base version based on its context and intended meaning (i.e "be" is the lemma for "was" and "is").

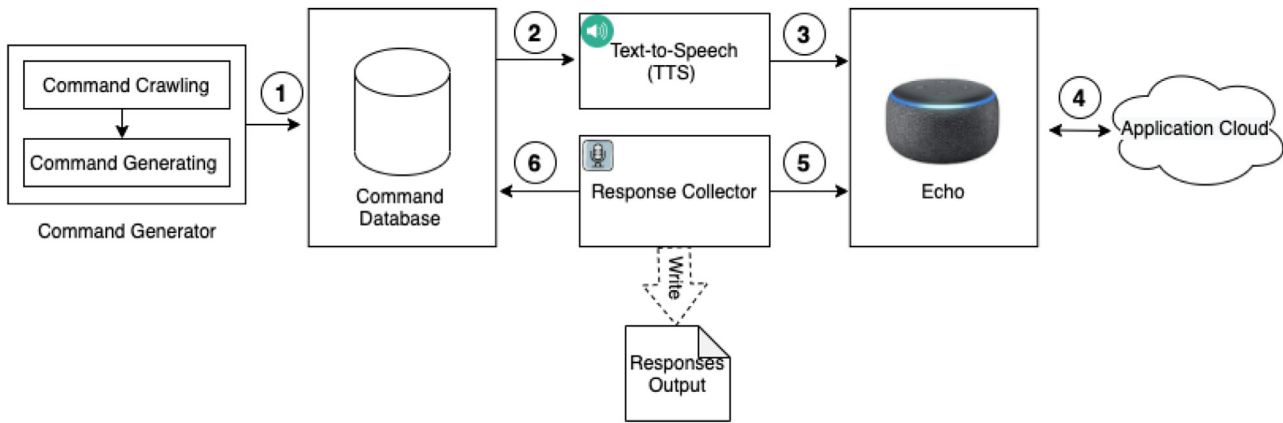


Fig. 6. Interaction tool framework.

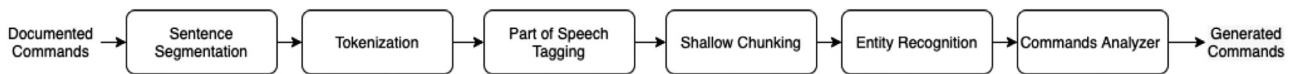


Fig. 7. Commands generator NLP process.

4.2. Prototype development

VORI builds the POS tagging, shallow chunking, and entity recognition using NLTK natural language toolkit, speech synthesis module using pyttsx3 a text-to-speech conversion library, speech recognition module using PyPI library, and rule-based matching using SpaCy an open-source software library for advanced Natural Language Processing. We ran a series of experiments by using VORI. We performed all the experiments using third-generation Echo and Apple MacBook with a 2.7GHz CPU and 16GB of RAM.

Text to Speech (TTS). This component is responsible for automating speaking a command to the smart speaker. We implement a speech synthesis module using Python library [29]. It takes text command as input and converts it to a synthesized voice. The result is robotic or artificial voices that transmit to the Alexa Echo device. Then Alexa invokes the target app and starts the interaction. The app provides an audio reply that is captured by the response collector component.

Response collector. This component is responsible for recording the app's responses and transforming them into texts by using the speech recognition module. We implement a speech recognition module using Python library [30] that captures responses, communicates with the Google API in real-time to convert speech to text, and produces a text file containing each app's responses as input for further analysis.

5. Evaluation

While VORI can be used to evaluate smart speaker apps in general, we center our evaluation on smart speaker apps that involve alarms and alerts used to help prompt users. The reason is that this kind of *reminder-type* apps cannot be tested using existing smart speaker text-based simulators.

Data set collection: There are 23 different categories and 66 sub-categories listed on the Alexa app store. The app store does not support filtering apps that make use of alarms and alerts. We wrote a Selenium-based [31] web crawler to automatically access an app web page and collect app information from the Alexa app store [26] under the "Alarms & Clocks", "Calendars & Reminders", and "To-Do Lists & Notes" lists provided by Amazon under the "Productivity" category to create the data set.

The web crawler automatically collect app information. For each app, we obtained the following information: the invocation name, cate-

Table 2

Basic statistics of number of apps in each category.

Category	Description	# Apps
Record Tracking	Activated by the use, i.e to-do List and Calendar	119
Reminders	Automatically activated, i.e. Alarm	8

Table 3

Computation Overhead (in second) of VORI.

Experiment	# Apps	# Commands	Time (Seconds)	Accuracy
Interactability	127	83584	5,896	98%
Recovery from failure	127	381	1,279	99.3%

gory, description of the app's capabilities, sample voice commands, and customer reviews. We restricted our scope to free applications that are in English, and with a customer reviews greater than five. This left us with a final list contains a total of 127 applications (Table 2). This list collected from the U.S. store on March 2021. We extracted 653 sample commands in total.

Accuracy of VORI: We first measure the accuracy of VORI to showcase the interactability score is a suitable metric. Regarding accuracy, we care about the classification accuracy of the responses analyzer. Accuracy of response prediction impact the interactability score. We randomly select 70 apps. Then we manually interact with them to collect as many responses as possible. 170 responses from apps are collected, including 70 recognized responses, 60 failure responses, and 40 help responses. Such collected responses use as the ground truth for comparison with the result explored by VORI. We found that VORI achieved good performance with 98% accuracy for predicting the *recognized* verbal commands that apps in Alexa can accept. An accuracy of 99.3% for predicting the *failure* responses. Finally, VORI achieved 100% accuracy in prediction *help* provided by apps to allow users to recover from failure easily.

Overhead of VORI: VORI is designed to analyze the interactability of apps. So the overhead of traversing is important to evaluate the performance of VORI. VORI has generated 82,931 commands within 6401 s. Table 3 shows the overall overhead of interactability testing with VORI. It has analyzed 127 apps and measure their interactability within 5896 s. Each app costs about 46 s on average, the time varies

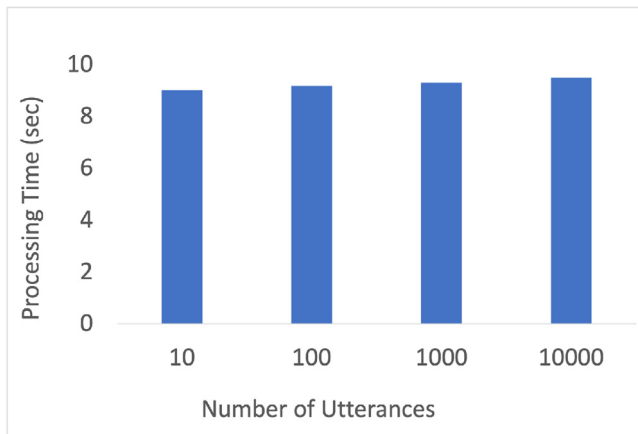


Fig. 8. Processing time of different number of utterances (the result averaged over ten trials).

for different categories that depends on the function and the branches of the app. Then we evaluate VORI’s performance in analyzing the help provided by apps to allow users to recover from error easily. To evaluate, we send three commands to each app that marked as a failure by the app from the previous test. In this process, VORI has analyzed 127 apps within 1279 s in total.

Overhead of the numbers utterances: We looked at the performance penalty, whether increasing the number of utterances impacts app performance and adds additional overhead to the request processing time ($\Theta_{input} < \Theta_{utterances}$). To answer the question, we build four apps that used 10, 100, 1000, and 10000. sample utterances to check the overhead of the bucket size. We query identical commands to all apps. Fig. 8 shows the processing overhead of increasing number of utterances. The evaluation results averaged over ten folds, each fold used different set of commands.

We found that the larger number of sample utterances *does not have a negative performance impact*. The performance from the speech send as input until the moment the app is called is Amazon’s responsibility. The utterance matching in Amazon works as follows. First, they take the app utterance file and parse the words and phrases into a decision tree. Then, when the speech is received, they will convert it to phonemes, and then into words. Alexa goes through the trees and finds the highest confidence matching. The preparation of utterance is done at submit time, while the speech analysis has to be done at run time. Therefore, the size of the utterance file doesn’t matter, it’s the depth and number of branches of the tree.

Finding: To compute *interact_score* (Eq. (1)) we used *generated commands* to investigate the flexibility of these apps measure by the percentage of commands they accept. For each app, we use VORI *command modification component* to generate corresponding commands from all 653 crawled lists of commands. In total, 82,931 commands generated. In this stage, we send the 82,931 commands as input to the target app, and we observe the app responses. For example, to add a record: "Alexa, ask [app’s name] to add lunch with friends on July 8th at 11:30 am", to look-up the active records: "Alexa, ask [app’s name] what is in my schedule July 8th?", to adjust records: "Alexa, ask [app’s name] to move my lunch with friends over to July 11th at 12 pm". Then VORI uses *response analysis component* to compute $\Theta_{utterances}$ using (Eq. (2)).

From Fig. 9, we can observe that 90 (70.8%) apps only accept at most 40% of the total commands in the dataset. Among them, 53 apps only accept a pretty strict input that exactly matches the developer’s predefined sample commands. For example, one app restricts the input entry to be: "remember my [to-do task] as" followed by only "this week" or "next week", no other input will be accepted. The remaining 37 apps accept up to 80% of the total commands in the dataset. These apps have better interactability and more flexible that can accept a free flow of

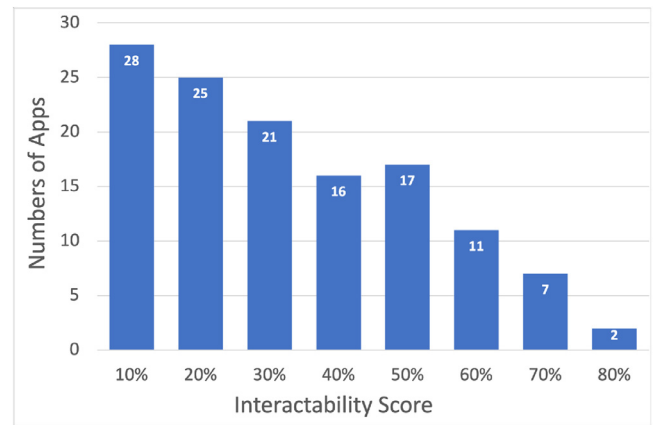


Fig. 9. Interactability Score.

Table 4

How easy it is for a user to recover from failure.

	#Apps	Situation
Support recovery from failure	19	Respond with a suggested command to use
Don’t support recovery	108	Respond with an error message then quit

text in any structure, giving the user more flexibility to add more information about the request. For example, the app accepts both following commands: "remind me to pick up kids from school at 3 pm", or "add pick up kids to my today reminder".

Then, we measure if the user uses an out-of-domain command that is not implemented by the developer how the app will behave, does the app responds with suggested commands, or just respond with the default error response and quit. For good interactability, the app is supposed to provide informative guidance such as instructions on how to use core functionality and the correct command. To evaluate, we use VORI to send three unrecognized commands to each app.

Even though, Amazon design guidelines suggest that every app should implement a "help" function for better user experiences. From Table 4, we can see most of the apps (108) provide informative instructions on how to use core functionality if the command *explicitly* asks for help. Among them, less than a third (19 apps) reply with the correct command if the input command does not recognize by the app. However, the rest 57 apps don’t provide a help function, they just respond with *failure* response and quit. So it is hard for the user to determine the cause of the error that may cause bad user experiences. Example of bad customer reviews "The app doesn’t work, don’t know why", "Very poor, doesn’t recognize the common command. disappointed", and "it doesn’t even understand that phrase".

Comparison of VORI against existing techniques: There are two main techniques to test the usability of VUI: user study, and an automated testing tool. **Traditional user study testing** efforts center on questionnaires and interviews to gauge the usability and utility of a given VUI. A large group of users studies focuses on understanding how users interact with smart speakers, and what challenges they face [8,11,13]. We categorized papers based on the size of the test group, evaluation matrix, and number of analyzed skills. The results are presented in Table 5

Some of the more common questionnaires for overall system design include the Mean Opinion Scale (MOS) [33], System Usability Scale (SUS) [34], and NASA Task Load Index (NASA-TLX) [35]. In addition to this, there are questionnaires focusing only on voice system usability, such as the Subjective Assessment of Speech System Interfaces (SASSI) [36,37], and Speech User Interface Service Quality (SU-ISQ) [38,39].

Table 5
Traditional user study usability testing.

Research Work	Size of Test Group	Skills Analyzed	Evaluation Matrix
[11]	12	1	What common obstacle categories users encountered, and what tactics they developed to overcome these obstacles
[13]	18	1	Completion time, number of errors, number of turns per task, and number of completed stages
[8]	114	Undefined	System Usability Scales and User Experience Questionnaire(UEQ) [32]

Limitations: First, a small-scale user study is unlikely to capture the diverse ways of communicating the same piece of information. A larger-scale user study, on the other hand, while able to yield better results, will be expensive to conduct for every app that utilizes a VUI. Second, the result is hard to interpret. It is difficult to use them to identify issues related to user interaction, such as determining whether their skill can adequately match what users' might say. VORI scale-up the analysis by developing automated tools.

More recently, there is an interest in building **automatic testing tools** to facilitate the analysis of skills on a large scale [21,22]. Skill-Explorer [21] is a grammar-rule-based testing tool to automatically explore the behaviors of skills to detect privacy violations. They tested 28,904 Amazon skills and 1,897 Google actions. They identified 1,141 skills request personal information without disclosing it in their privacy policies. While VerHealth [22] proposed a testing tool to investigate the health-related skills on Alexa and how well they comply with existing privacy and safety policies. The authors tested 813 health-related skills on the Amazon Alexa platform, and adopts a simple interaction model to collect limited responses from skills.

Limitations: These initial efforts all rely on the simulator tool developed by Amazon for their Alexa platform that allows developers to communicate with a skill using texts and observe its outputs also in text form. However, this text-based simulator cannot be used to test audio aspects such as accents and pronouncements, and is restricted to certain APIs. Skills that involve some type of alarms, for instance, cannot be tested using the simulator. SkillExplorer only focuses on identifying skills that collect private information, without evaluating skills usability. VerHealth mainly focuses on detecting health-related policies violations. These tools provide no feedback to developers on how well the system will perform in the wild, such as speaking cadence and accent for audio systems. While VORI focuses on usability metric by quantifying how well well a VUI is able to predict what commands an end user will use to interact with the interface.

6. Additional discussion and conclusion

In the future, we would like to expand the VORI tool to operate well beyond the Alexa context and perform a cross-platform analysis. VORI tool can be easily extended to other smart speaker platforms since smart speakers generally all have similar system architecture, support similar interaction interface, follow very similar patterns for building apps, and follows similar patterns of voice commands. We also want to consider, the newer paradigms of VUI, such as Firefox Voice [40] and the DIY Assistant [41], since they are open-source web-based voice assistants.

We would like to explore more about the usability of human-to-app interaction. For that, we need people trying commands and trying each app. Investigate the correlation between VORI and the usability tests with humans. Even though VORI can also help VUIs developers to proactively test their own apps during the development phase to evaluate interactability, more research is needed to explore the design space to further improve the interactability.

Currently, VORI is designed to evaluate one usability metric (interactability) for VUIs. We are aware there are other usability metrics, such as measuring users' overhead to accomplish a task without much confusion or undue effort. Our future work will focus on extending our tool to check other metrics and evaluate a broader set of usability metrics.

To conclude, in this paper we developed a tool called VORI, a framework that can systematically evaluate the interactability of VUI apps. The key technique of collecting input commands that reflects what users will say in the real world is aggregating the sample commands from similar apps made by multiple developers. Then, a suite of NLP processes is applied to these commands to derive larger test set commands and allow cross-app evaluation. Finally, VORI analyzes apps' responses, classify them into different types, and further compute interactability score according to their responses. We applied VORI to analyze 127 applications on Alexa by sending over 82,931 commands. Our analysts show there are still many open challenges to build usable VUI, to improve the user experience of VUIs. The app developer still needs to accurately determine how a user will actually talk to the app and configure the list of ways a user could invoke various app functions.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

- [1] Amazon, <https://alexa.amazon.com/>.
- [2] Google, https://store.google.com/product/google_home.
- [3] L. accessed 24-Jan-2021, <https://www.fool.com/investing/2020/01/24/60-million-americans-now-own-smart-speakers.aspx>.
- [4] L. accessed 24-Jan-2021, <https://techcrunch.com/2019/01/04/more-than-100-million-alexa-devices-have-been-sold/>.
- [5] voicebot.ai, (<https://voicebot.ai/2019/02/15/google-assistant-actions-total-4253-in-january-2019-up-2-5x-in-past-year-but-7-5-the-total-number-alexa-skills-in-u-s/>). Accessed: 2020-11-20.
- [6] D. Kumar, R. Paccagnella, P. Murley, E. Hennenfent, J. Mason, A. Bates, M. Bailey, Skill squatting attacks on amazon alexa, in: Proceedings of the 27th (USENIX) Security Symposium ((USENIX) Security 18), 2018, pp. 33–47.
- [7] J. Kowalski, A. Jaskulska, K. Skorupska, K. Abramczuk, C. Biele, W. Kopeć, K. Marasek, Older adults and voice interaction: a pilot study with Google home, in: Proceedings of the Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, 2019, pp. 1–6.
- [8] A. Pyae, T.N. Joelson, Investigating the usability and user experiences of voice user interface: a case of google home smart speaker, in: Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct, 2018, pp. 127–131.
- [9] B.R. Cowan, P. Doyle, J. Edwards, D. Garaialde, A. Hayes-Brady, H.P. Branigan, J. Cabral, L. Clark, What's in an accent? the impact of accented synthetic speech on lexical choice in human-machine dialogue, in: Proceedings of the 1st International Conference on Conversational User Interfaces, 2019, pp. 1–8.
- [10] D. Pal, C. Arpnikanondt, S. Funilkul, V. Varadarajan, User experience with smart voice assistants: the accent perspective, in: Proceedings of the 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE, 2019, pp. 1–6.
- [11] C. Myers, A. Furqan, J. Nebolsky, K. Caro, J. Zhu, Patterns for how users overcome obstacles in voice user interfaces, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, 2018, pp. 1–7.
- [12] A. Purington, J.G. Taft, S. Sannon, N.N. Bazarova, S.H. Taylor, " alexa is my new bf" social roles, user satisfaction, and personification of the amazon echo, in: Proceedings of the 2017 CHI conference extended abstracts on human factors in computing systems, 2017, pp. 2853–2859.
- [13] P. Kirschthaler, M. Porcheron, J.E. Fischer, What can I say? Effects of discoverability in vuis on task performance and user experience, in: Proceedings of the 2nd Conference on Conversational User Interfaces, 2020, pp. 1–9.
- [14] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, B. Strope, "your word is my command: Google search by voice: a case study, in: Advances in Speech Recognition, Springer, 2010, pp. 61–90.
- [15] E. Corbett, A. Weber, What can I say? Addressing user experience challenges of a mobile voice user interface for accessibility, in: Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services, 2016, pp. 72–82.

- [16] A. Furqan, C. Myers, J. Zhu, Learnability through adaptive discovery tools in voice user interfaces, in: Proceedings of the CHI Conference Extended Abstracts on Human Factors in Computing Systems, 2017, pp. 1617–1623.
- [17] C. Lentzsch, S.J. Shah, B. Andow, M. Degeling, A. Das, W. Enck, Hey Alexa, is this skill safe?: Taking a closer look at the alexa skill ecosystem, Proceedings of the Network and Distributed Systems Security (NDSS) Symposium 2021 (2021).
- [18] A.S. Alrumayh, S.M. Lehman, C.C. Tan, Understanding and mitigating privacy leaks from third-party smart speaker apps, in: Proceedings of the IEEE Conference on Communications and Network Security (CNS), IEEE, 2021, pp. 1–9.
- [19] S. Liao, C. Wilson, L. Cheng, H. Hu, H. Deng, Measuring the effectiveness of privacy policies for voice assistant applications, in: Proceedings of the Annual Computer Security Applications Conference, 2020, pp. 856–869.
- [20] L. Cheng, C. Wilson, S. Liao, J. Young, D. Dong, H. Hu, Dangerous skills got certified: measuring the trustworthiness of skill certification in voice personal assistant platforms, in: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 1699–1716.
- [21] Z. Guo, Z. Lin, P. Li, K. Chen, Skillexplorer: understanding the behavior of skills in large scale, in: Proceedings of the 29th {USENIX} Security Symposium ({USENIX} Security 20), 2020, pp. 2649–2666.
- [22] F.H. Shezan, H. Hu, G. Wang, Y. Tian, Verhealth: vetting medical voice applications through policy enforcement, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 4 (4) (2020) 1–21.
- [23] A.C. Morris, V. Maier, P. Green, From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition, in: Proceedings of the 8th International Conference on Spoken Language Processing, 2004.
- [24] A.S. Alrumayh, S.M. Lehman, C.C. Tan, Emerging mobile apps: Challenges and open problems, CCF Trans. Pervasive Comput. Interact. 3 (1) (2021) 57–75.
- [25] Best practices for sample utterances and custom slot type values, <https://developer.amazon.com/en-US/docs/alexa/custom-skills/best-practices-for-sample-utterances-and-custom-slot-type-values.html>. Accessed: 2021-06-8XS.
- [26] Alexa skills store, (<https://www.amazon.com/alexa-skills/b>) Accessed: 2020-08-20.
- [27] Natural language understanding, (<https://towardsdatascience.com/how-amazon-alexa-works-your-guide-to-natural-language-processing-ai-7506004709d3>). Accessed: 2021-03-25.
- [28] Developing your first skill, (<https://developer.amazon.com/en-US/docs/alexa/alexa-skills-kit-sdk-for-nodejs/develop-your-first-skill.html>), Accessed: 2021-03-01.
- [29] Text to speech (tts) library for python, (<https://pypi.org/project/pyttsx3/>), Accessed: 2021-03-01.
- [30] Speech recognition library for python, (<https://pypi.org/project/SpeechRecognition/>), Accessed: 2021-03-01.
- [31] Selenium, (<https://selenium-python.readthedocs.io>), Accessed: 2020-08-20.
- [32] M. Schrepp, A. Hinderks, J. Thomaschewski, Design and evaluation of a short version of the user experience questionnaire (UEQ-S), Int. J. Interact. Multimed. Artif. Intell. 4 (6) (2017) 103–108.
- [33] M.D. Polkosky, J.R. Lewis, Expanding the mos: Development and psychometric evaluation of the mos-r and mos-x, Int. J. Speech Technol. 6 (2) (2003) 161–182.
- [34] D. Ghosh, P.S. Foong, S. Zhang, S. Zhao, Assessing the utility of the system usability scale for evaluating voice-based user interfaces, in: Proceedings of the 6th International Symposium of Chinese CHI, 2018, pp. 11–15.
- [35] Tlx @ nasa ames - home, (<https://humansystems.arc.nasa.gov/groups/TLX/>), Accessed: 2020-07-23.
- [36] K. Hone, Usability measurement for speech systems: Sassi revisited, in: Proceedings of the SIGCHI Conference Paper, Toronto, 2014.
- [37] K.S. Hone, R. Graham, Towards a tool for the subjective assessment of speech system interfaces (SASSI) (2000).
- [38] M.D. Polkosky, Toward a social-cognitive psychology of speech technology: affective responses to speech-based e-service (2005).
- [39] M.D. Polkosky, Machines as mediators: the challenge of technology for interpersonal communication theory and research, in: Mediated Interpersonal Communication, Routledge, 2008, pp. 48–71.
- [40] J. Cambre, A.C. Williams, A. Razi, I. Bicking, A. Wallin, J. Tsai, C. Kulkarni, J. Kaye, Firefox voice: an open and extensible voice assistant built upon the web, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, 2021, pp. 1–18.
- [41] M.H. Fischer, G. Campagna, E. Choi, M.S. Lam, Diy assistant: a multi-modal end-user programmable virtual assistant, in: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2021, pp. 312–327.