Automatically Generating and Improving Voice Command Interface from Operation Sequences on Smartphones

Lihang Pan
plh18@mails.tsinghua.edu.cn
Department of Computer science and Technology, Tsinghua University
Beijing, China

Chun Yu*
chunyu@mail.tsinghua.edu.cn
Department of Computer science and Technology, Tsinghua University
Beijing, China

JiaHui Li
ljh1304607285@hotmail.com
Department of Computer science and Technology, Tsinghua University
Beijing, China

Tian Huang
any3231@126.com
Department of Computer science and Technology, Tsinghua University
Beijing, China

Xiaojun Bi
xiaojun@cs.stonybrook.edu
Department of Computer Science, Stony Brook University
United States

Yuanchun Shi
shiyc@tsinghua.edu.cn
Department of Computer science and Technology, Tsinghua University
Beijing, China

ABSTRACT

Using voice commands to automate smartphone tasks (e.g., making a video call) can effectively augment the interactivity of numerous mobile apps. However, creating voice command interfaces requires a tremendous amount of effort in labeling and compiling the graphical user interface (GUI) and the utterance data. In this paper, we propose AutoVCI, a novel approach to automatically generate voice command interface (VCI) from smartphone operation sequences. The generated voice command interface has two distinct features. First, it automatically maps a voice command to GUI operations and fills in parameters accordingly, leveraging the GUI data instead of corpus or hand-written rules. Second, it launches a complementary Q&A dialogue to confirm the intention in case of ambiguity. In addition, the generated voice command interface can learn and evolve from user interactions. It accumulates the history command understanding results to annotate the user's input and improve its semantic understanding ability. We implemented this approach on Android devices and conducted a two-phase user study with 16 and 67 participants in each phase. Experimental results of the study demonstrated the practical feasibility of AutoVCI.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; **Ubiquitous and mobile computing systems and tools**.

KEYWORDS

operation sequence, generation system, voice command interface, interaction-centered nature language understanding

*indicates the corresponding author.

ACM Reference Format:
Lihang Pan, Chun Yu, JiaHui Li, Tian Huang, Xiaojun Bi, and Yuanchun Shi. 2022. Automatically Generating and Improving Voice Command Interface from Operation Sequences on Smartphones. In *CHI '22: The ACM CHI Conference on Human Factors in Computing Systems, April 30 – May 6 2022, New Orleans, LA.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3491102.3517459

1 INTRODUCTION

Voice interaction has many advantages, such as rich expressions, low learning cost, and support for hands-free and eyes-free interactions [62, 68]. Using voice commands to automate smartphone tasks can effectively improve the interactivity of mobile apps. However, it is difficult to create task-oriented voice user interfaces (VUIs) for various applications. One of the most significant challenges is to understand flexible natural language commands accurately. The developers need to collect numerous corpus [10], create hand-write rules [9], or train machine learning models [63]. Moreover, the development efforts will grow significantly as the number of tasks increases. As a result, only a small number of tasks are equipped with voice interfaces. However, many users, especially non-programmers, have a strong desire to create voice interfaces for various tasks [34, 36].

In this paper, we proposed AutoVCI, an original approach to automatically generate and improve a voice command interface (VCI) from smartphone operation sequences, aiming at reducing the overhead of constructing VCIs. As shown in Figure 1, the VCI designers only need to provide a sequence of touch-based operations for accomplishing a task (i.e., how a task will be executed through touch operations). AutoVCI automatically generates a VCI based on the operation sequences through the following steps. (Intention Recognition) The generated VCI maps a voice command to a task (e.g., the user wants to make a video call) leveraging semantic similarities between the command and the operation sequences. (Parameter Identification) It then identifies parameters (e.g., the recipient is *Alice*) from the command utilizing similarities between the command and run-time GUI. The semantic understanding does not require any corpus, models or hand-written rules. (Complementary Dialogue) If unexpected ambiguity occurs, the VCI launches an complementary Q&A dialogue to maintain the accuracy of the command understanding. We carefully designed the questions so that they are easy to answer, and minimized the number of questions via an information entropy based method. (Semantic Accumulation) The generated VCI also accumulates semantics and evolves from interactions: it leverages the history command understanding results to annotate new user commands, and improves its semantic understanding ability. The accumulation reduces (in some cases even avoids) asking the users additional questions for subsequent commands.

To evaluate the performance of AutoVCI, we implemented it on an Android device and provided a tool that automatically collected operation sequences from the smartphone. We generated a VCI supporting 45 tasks from 11 applications. A user study ($N = 16$) showed that users were able to automate GUI tasks via the VCI with an overall success rate of 98.4%. In addition, semantic accumulation reduced the interaction burden caused by extra dialogues and enhanced the usability of AutoVCI. During the experiment, the average number of extra dialogue rounds decreased from 2.1 to 0.7, and more than 40% of the commands from history could be reused and executed directly. Users gave positive feedback and commented that they could answer the complementary questions easily. To investigate AutoVCI's ability of continuous semantic learning at a larger scale, we further expanded the user study to a second phase ($N = 67$, conducted online). At the end of Phase II, more than 70% of the commands could be directly triggered, and the extra dialogues were only 0.4 rounds on average. Subsequently, we conducted an offline evaluation on the data collected in the user study, which measured different components and explained the good performance of AutoVCI.

In the remainder of this paper, we first summarize relate works on constructing voice user interfaces. We then introduce a use case scenario to clarify how the VCI is generated and cumulatively self-enhanced. After that, we describe the details of AutoVCI design, including VCI generation, intention recognition, parameter identification during execution, and semantic accumulation. Subsequently, we demonstrate the feasibility and the self-improvement ability of AutoVCI through a user study and an offline evaluation. Finally, we discuss the implications and possible directions for future work.

2 RELATED WORK

The construction of task-oriented voice user interfaces (VUI) consists of two parts [38]: 1) the configuration of task properties, including the required parameters (for example, *sending a video call* requires a parameter for the receiver) and execution scripts (how to send a video call automatically), and 2) mapping verbal commands to the GUI tasks. The generated VUIs may evolve while interacting with the users, which can be regarded as an extension of the generation process. This section presented related works in task property configurations and user command understanding, and discussed how AutoVCI contributes.

2.1 Task Property Configurations

Task properties are usually configured manually by VUI developers through programming [33, 52]. Existing commercial voice interfaces (for example, Siri) provide adequate application programming interfaces (APIs) to support developers in adding new tasks. Some existing systems apply end-user programming [27] to infer task properties from interaction histories [7, 33] or verbal commands [1, 10, 27, 36, 37, 47]. In addition to programming by computer languages, previous works [11, 18] calculated properties from specific GUI annotations. The task execution of AutoVCI is built on top of existing programming by demonstration (PBD) systems. Programming by demonstration (PBD) [20, 56] is a widely used end-user programming approach, in which the system automatically generates runnable scripts after users demonstrate the task execution process in the original GUI [44]. However, existing PBD systems rely on extra inputs to extract parametric operations. Sugilite [34] and VASTA [54] require natural language descriptions for the demonstration and determine task parameters via text comparison. Kite [38] provides a visualization platform that requires additional configuration to generate dialogue templates. In contrast, AutoVCI leverages interaction sequences to infer task properties accurately and does not require additional inputs, which enhances the scalability of AutoVCI.

2.2 User Command Understanding

One of the major contributions of AutoVCI is on user command understanding. Existing strategies rely only on static natural language processing (NLP) algorithms. AutoVCI utilizes the rich semantic information contained in the GUI instead of any corpus, rules, or programming. When encountering ambiguous instructions, AutoVCI launches additional dialogues to guarantee understanding accuracy. Accurate command understanding with little development effort is a fundamental feature of AutoVCI, and a significant difference and improvement comparing with other existing PBD systems.

2.2.1 Existing Natural Language Command Understanding Approach. Existing voice interfaces rely heavily on static NLP algorithms to understand user commands. The most traditional approach is to match user commands with regular expressions [9, 68] or hand-written rules [22, 60]. Some systems employ more sophisticated algorithms, such as decision trees [8], word dependencies [54], and existing semantic understanding frameworks [57]. All these approaches require significant development efforts [38]. Nevertheless, they can not adapt well to the dynamics of natural language commands [10, 40, 63] because their capabilities have already been determined once interface generation finished. Taking the commonly used Combinatory Categorial Grammar (CCG) [10, 34, 36, 37, 58] algorithm as an example, the error rate for intention recognition is 15.4% and that for parameter identification is 5.4% in a system that supports only 13 tasks [10].

Some existing algorithms select the most similar task to the instruction as the semantic understanding result. The similarities can be calculated in many ways, such as n-gram [22, 31] and word embedding [54]. These approaches require only a tiny amount of corpus and can flexibly handle patterns that did not occur before. However, lacking the mechanism of user input and confirmation, their results are not always adequately accurate [54].

With the development of artificial intelligence, many systems use deep neural networks that could achieve good performance in general [50, 53]. But developers need to collect vast amounts [16, 21] of training corpus and carefully refine the network structure

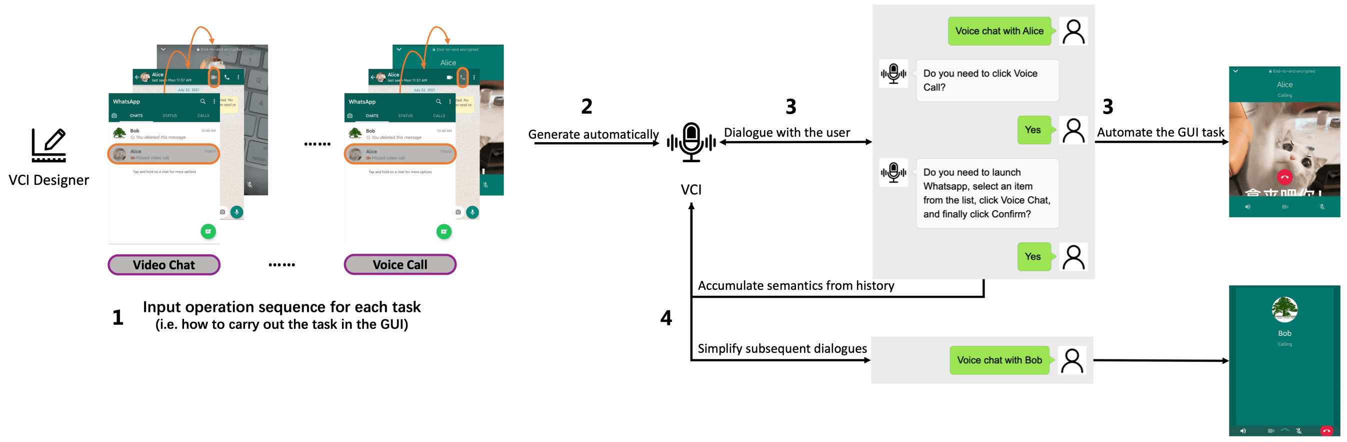

VCI Designer
Video Chat
Voice Call
1 Input operation sequence for each task (i.e. how to carry out the task in the GUI)
2 Generate automatically
VCI
3 Dialogue with the user
Voice chat with Alice
Do you need to click Voice Call?
Yes
Do you need to launch Whatsapp, select an item from the list, click Voice Chat, and finally click Confirm?
Yes
3 Automate the GUI task
Accumulate semantics from history
4 Simplify subsequent dialogues
Voice chat with Bob

Figure 1: How is the VCI generated and cumulatively self-enhanced? 1) the VCI designer inputs an operation sequence for each task; 2) AutoVCI generates VCI automatically; 3) the generated VCI interprets user's commands with potential additional dialogues and automates the task; 4) the VCI accumulates semantics and simplifies dialogues in subsequent user interactions.

and parameters [67] in order to achieve better recognition results, which increases the development cost significantly.

2.2.2 Strategies for Ambiguous Commands. In often cases, user commands may be understood inaccurately, which leads to conversation breakdowns [12] due to the limitations of existing NLP algorithms [25]. Users have to memorize the voice command patterns supported by the voice interfaces [29, 48], which decreases users' satisfaction and willingness of using the system [28, 40, 65].

Few dialogue systems provide effective solutions to unmanageable commands as they may introduce extra semantic understanding errors [35, 37]. Existing strategies suffer from various problems and are not widely used. Although developers can fix the breakdowns programmatically [15, 30], that would further increase the workload in implementing the voice interface. *Thomason* [58] determines user intentions or parameters with natural language questions based on inaccurate semantic understanding results, which is not suitable for cold-starting systems as the beforehand data do not always exist. SOVITE [35] explores how breakdowns can be fixed effectively. However, it relies on visual feedback and requires the user to operate on the GUI. This approach requires additional modality switches, which may be inconvenient or even impossible.

2.2.3 Learning for Understanding. Learning user preferences is a popular topic, which has been well-studied for recommendation systems [41, 59]. In the field of human-computer interaction, some interaction systems can improve themselves while interacting with users. *Rana* [51] assists users to navigate indoors by determining their short-term preferences through dialogues, while *Gervasio* [23] and *Krzywicki* [32] assist users to make schedules.

However, learning how to understand user commands has rarely been proposed in interaction systems. In *Thomason* 's article [58], users can teach the system synonyms of existing concepts by inefficient trials and errors. Learning from users is an essential capability, allowing for persistent knowledge storage and iterative updates. In AutoVCI, we successfully improved the semantic understanding ability of the generated voice interface according to this principle.

2.3 Generating voice interfaces from GUIs

Previous work has generated voice interfaces by mapping voice commands to commands in an existing GUI interface, enabling speech-based access to contents in Wikipedia [55], email [64], and in-vehicle information applications [46]. These papers demonstrated many advantages of voice interfaces, such as high efficiency [55] and support for various situations [46, 55]. Some prior work has shown that this approach can lead to poor usability, because it is challenging for voice interfaces to present complex information (such as long lists) which can occur in GUIs [55, 64]. AutoVCI avoids this problem since it is focused on executing user commands, and not on verbally presenting GUI contents. Verbal command understanding is the key challenge that all voice interfaces must address. For example, SpeechActs [64] searched keywords in the commands to recognize the intentions. However, the performance of this method was minimal. AutoVCI aims at understanding user commands accurately at a low cost.

3 A USE CASE SCENARIO

In this section, we use an example to illustrate how to generate a VCI via AutoVCI and how the generated VCI is used. This example demonstrates how the interfaces generated by AutoVCI could improve accessibility, and is one of the many possible use cases.

Alice, as a lay person without any programming background, generated a VCI through AutoVCI to improve accessibility for the elderly. The generated VCI only requires voice input and output, and it can be used in many applicable scenarios of voice interaction. Alice's parents are in their 70s and have difficulties using their smartphones. Voice assistants would be helpful, but the existing ones only support a small percentage of smartphone tasks. For example, making a video call with Alice using Facebook Messenger, which Google Assistant or Siri is not supported. Alice decided to create a VCI to support frequent tasks for her parents. She recorded the operation sequences for those tasks, including making a video call in Messenger, as shown in Figure 2. The operation sequences were uploaded to our server, and AutoVCI automatically generated a VCI.

Alice shared the generated VCI with her husband. He activated the VCI and gave a verbal command, *Video Call Mum*. The newly generated VCI could not understand the command and started an additional dialogue asking whether the operation sequence contained *Whatsapp* and *Video Call*, as shown in Figure 3(A). It then understood the command according to the answers (Figure 3(A-3) & Figure 3(A-5)). After the confirmation from the user (Figure 3(A-6)), the VCI simulated touch events on the GUI and automated the task.

Alice's parents installed the VCI and gave the command *Send a video call to Alice*. The VCI could understand the command directly as it had already accumulated semantics from the dialogue with their son-in-law. As shown in Figure 3(B), the VCI only asked for confirmation before carrying out the task. Moreover, the VCI learned semantics from the simplified dialogue. Similar commands like *Send a video call to Coral* can be executed directly without any extra interactions in the future (Figure 3(C)).

4 THE DESIGN OF AUTOVCI

We proposed three design goals: 1) a designer can build a VCI supporting any GUI tasks with little effort; 2) user commands can be understood and executed accurately; 3) the interaction burden for end-users is low.

AutoVCI contains four phases: VCI generation, intention recognition, execution & parameter identification, and semantic accumulation, as shown in Figure 4. During VCI generation, the VCI designer only provides operation sequences for the supported GUI tasks. AutoVCI calculates the minimum semantic metadata with which the constructed VCI can recognize user intentions and fill in parameters. When the end-user gives a verbal command, the VCI first recognizes the intention. It then automates task execution by simulating touch events and determines parameter values simultaneously. The end-user may need to answer additional questions to guarantee semantic understanding accuracy. We carefully designed the questions to be easy to answer and minimized their quantity with an information entropy based theory. After finishing the execution, the VCI starts semantic accumulation, leveraging history semantic understanding results to update semantic metadata. In this way, it improves the semantic understanding ability and simplifies or even avoids extra interactions for subsequent commands. Table 1 shows an overview of task semantics used in AutoVCI.

4.1 VCI Generation

4.1.1 Inputs: Operation Sequences. To generate a VCI, the designers need to (1) choose the tasks that they want to create a VCI for and (2) import corresponding operation sequences to AutoVCI. We do not discuss the former because it varies between circumstances and needs. However, reducing VCI construction workload encourages designers to cover more tasks.

The operation sequences indicate how the smartphone users carry out tasks via the touch screen, as shown in Figure 5. The operations encode the execution logic and the task semantics, which AutoVCI extracts to construct a VCI automatically. An operation contains information about the operation element and the operation type. The text-input operation also contains the input text value. An operation element further contains information about its location, size, text, description, and child elements.

We provide a tool to facilitate designers to collect the sequences, with which they only need to carry out the tasks in the GUI manually. The data collection tool records the operation sequences and uploads them to the server. Data recording starts after the VCI designer clicks the floating button and ends when clicking it again. The tool creates a transparent floating window covering the original application window to intercept MotionEvents[5]. When encountering a MotionEvent, AutoVCI infers the operation element and type from the event attributes, uses Android's Accessibility Service[2] to record the GUI layouts and uses MediaProjection[4] to obtain the screenshots. The recorded layout and screenshot present the state before the recorded operation as well as the resulting state after the last operation. After logging all data, the tool simulates the intercepted event back to the GUI. The tool saves the sequences in JSON format and uploads them to AutoVCI. Designers can check and edit the sequences on a website if necessary. For example, they can manually delete or replace private data, as discussed in section 7.3.

With AutoVCI, anyone can generate voice interfaces without programming or gathering a data set to train a recognizer for the commands. The small amount of data required is very intuitive and easy to obtain [38]. AutoVCI significantly reduces the difficulties and workload of voice interface construction and encourages VCI designers to cover more smartphone tasks.

4.1.2 Semantic Calculations. AutoVCI calculates the minimum semantics required for command understanding and execution during the VCI generation: parametric operations and semantic vectors. The VCI generation finishes after the calculations, and end-users can give verbal commands to trigger tasks via the VCI.

Parametric Operation Identification. AutoVCI identifies which operations contain parameters and the types of the parameters. Parameters are the variable parts of the tasks. For example, in sending a message, the receiver and content (operation 1 & 2 in Figure 5) vary for different people at different times. Smartphone users determine the parameter values on their actual needs [34, 39], rather than repeating those in the previous operation sequences. The last operation in Figure 5 (clicking *send*) does not change in any case, meaning that it does not contain parameters.

We classified the parameters into two types. The first is the *list parameter*. For example, the user selects one friend from the contact list (operation 1 in Figure 5). The second is the *text parameter*, to which the user can assign any value via text entry, such as operation 2 in Figure 5. The inherent difference between the two types is that the GUI provides candidate values for *list parameters* while not for *text parameters*.

We used a heuristic algorithm to identify and classify the parameters. If the operation enters text into an *EditText*, it will be recognized as containing a *text parameter*. Otherwise, it will be recognized as containing a *list parameter* if satisfying all of the following conditions:

(1) The operation element is inside a list.
(2) The application developer does not specify a translation for the element's text to support multi-language. We look up the text in the *strings.xml*[6] extracted from the Android APK;

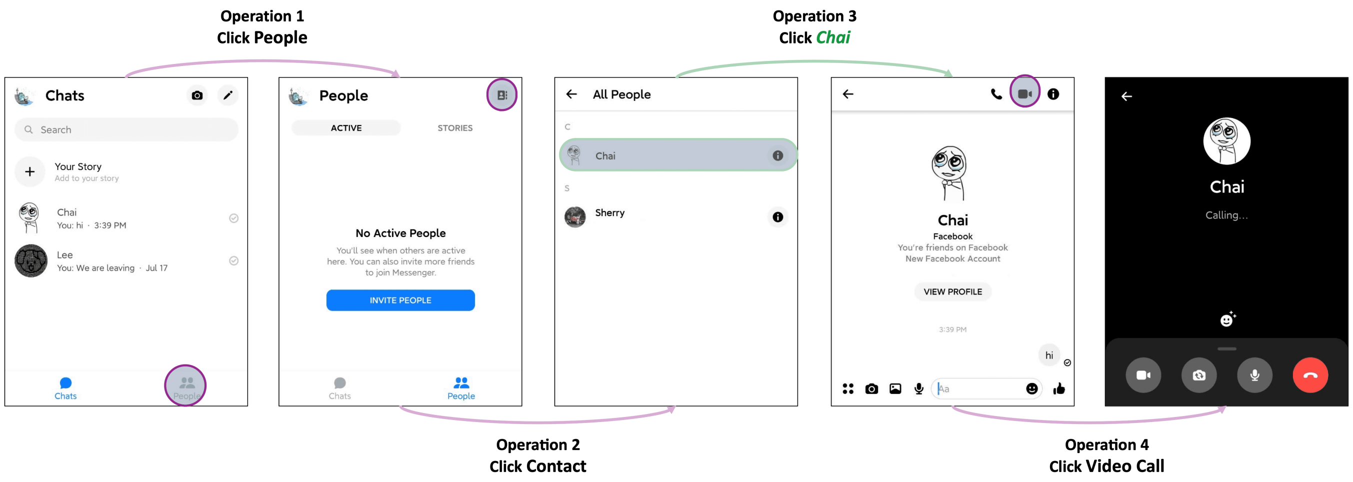
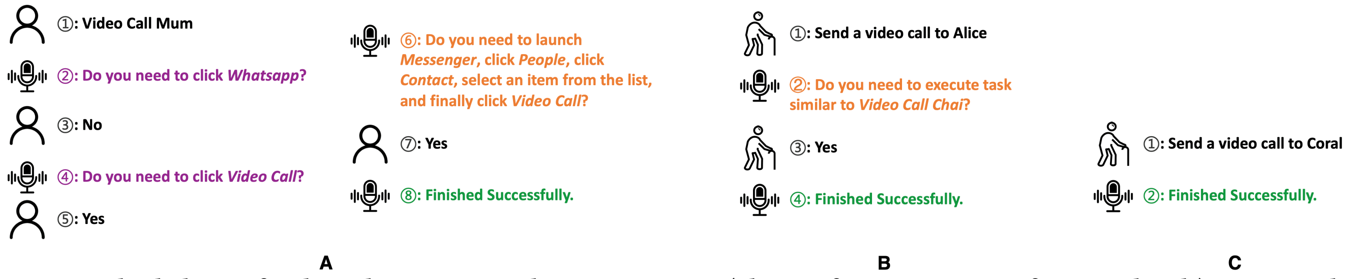


Figure 2: The operation sequence of the task *making a video call.*


①: Video Call Mum

②: Do you need to click *Whatsapp*?

③: No

④: Do you need to click *Video Call*?

⑤: Yes

⑥: Do you need to launch *Messenger*, click *People*, click *Contact*, select an item from the list, and finally click *Video Call*?

⑦: Yes

⑧: Finished Successfully.

A

①: Send a video call to Alice

②: Do you need to execute task similar to *Video Call Chai*?

③: Yes

④: Finished Successfully.

B

①: Send a video call to Coral

②: Finished Successfully.

C

Figure 3: The dialogues for the task in Figure 2. Filtration questions (please refer to section 4.2.3 for more details) are in purple and confirmation questions (details in section 4.2.4) are in orange. Part A is the dialogue between the VCI and Alice's husband. Part B and C are the dialogues between the VCI and Alice's parents, which are simplified compared with part A.

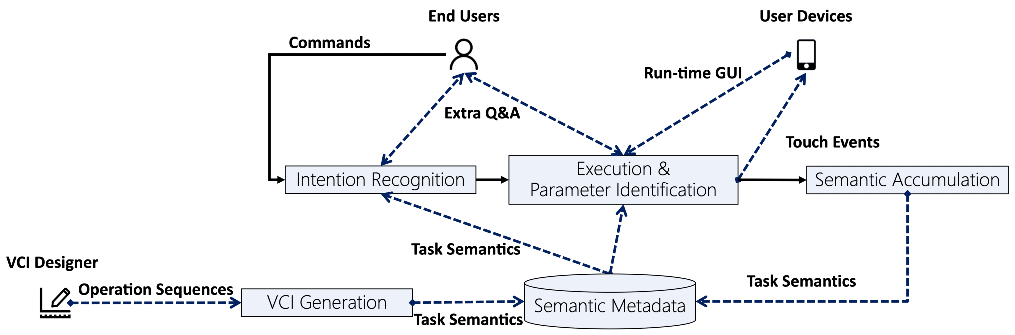


Figure 4: The architecture of AutoVCI. The solid arrows indicate how the commands are processed. The dashed arrows indicate data flows among different modules.

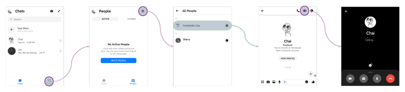
(3) The text of the operation element is long, or there are few results of searching the text with Google, or the text is a named entity (name of people or institutes, number, time, etc.)

The designer can correct the results manually if the algorithm makes any mistake.

Semantic Vector Calculation. AutoVCI calculates a semantic vector for each task. It extracts all texts[1] from non-parametric operations and calculates the vectors using a pre-trained BERT[21] model. As shown in Figure 6, a task semantic vector is composed of a word vector, a sentence vector, and a Bag-of-Words (BOW) [66] vector. The word vector summarizes the semantics of the texts, while the sentence vector describes the relationships among the texts. The BOW vector explicitly indicates the words existing in the operation sequence, bridging the gap between the sequence and the user commands with the common terms. The semantic vectors are used to calculate the similarities between tasks and user commands, as discussed in 4.2.2.

[1]for elements without texts, such as icons, we use content descriptions [3] instead.

Table 1: Semantics for the task *starting a video call with someone.*

Name	Explanation & Example	Calculated in	Used in
Operation Sequence	**How to carry out the task in the GUI** 1. Launch *Messenger* 2. Click *People* 3. Click *Contact* 4. Click *Chai* 5. Click *Video Call*	VCI Inputs (4.1.1)	VCI Generation (4.1.2) Result Confirmation (4.2.4) Task Execution (4.3.1)
Parametric Operations	**Variable parts in the sequence** 4. Click *Chai* $\Longrightarrow$ *List Parameter*	Operation Sequence (4.1.2)	Parameter Identification (4.3.2)
Semantic Vector	**A vector calculated from the texts in the sequence** *Messenger - People - Contact - Video Call* $\Longrightarrow$ 🟩🟩🟩🟩🟩🟪🟪🟪🟪🟪🟧🟧🟧🟧🟧🟧🟧	Operation Sequence (4.1.2) Semantic Accumulation (4.4.2)	Similarity Calculation (4.2.2)
Command Vector	**A vector calculated from the words in the command** Video call to Lee $\Longrightarrow$ 🟩🟩🟩🟩🟩🟦🟦🟦🟦🟦🟨🟨🟨🟨🟨🟨🟨	User Commands	Similarity Calculation (4.2.2) Result Confirmation (4.2.4)
BOW Embedding	**A vector indicating words in the sequence** (*Messenger, People, Contact, Video Call*): 1 others: 0	Operation Sequence (4.2.3)	Intention Filtration (4.2.3)
Extra Q&A	**Extra questions and answers** Q: Do you need to click *Voice Call*? A: Yes.	End-users	Filtration Questions (4.2.3) Confirmation Questions (4.2.4) Parameter Questions (4.3.2)
Run-time GUI	**Contents in user devices when executing the task** 	End-users' Devices	Parameter Identification (4.3.2)
Templates	**Regular expressions for the commands of the task** Video call to <>	Semantic Accumulation (4.4.1)	Template Matching (4.2.1) Result Confirmation (4.2.4)
Template Embedding	**A vector calculated from the words in the template** Video call to <> $\Longrightarrow$ 🟩🟩🟩🟩🟩🟦🟦🟦🟦🟦🟧🟧🟧🟧🟧🟧🟧	Semantic Accumulation (4.4.2)	Result Confirmation (4.2.4) Semantic Accumulation (4.4.2)

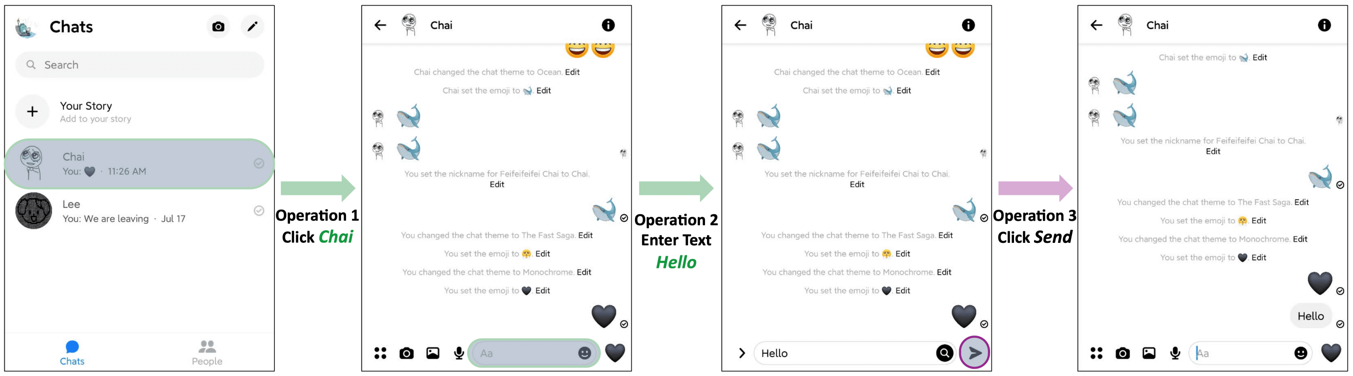


Figure 5: The operation sequence of the task *sending message in Facebook Messenger*. The operations in green (1: selecting receiver; 2 editing message contents) contain parameters, meaning that they can be different during actual execution. The operation elements are shaded in grey.

4.1.3 The Generated VCI. The generated VCI is composed of two components: the remote server and the VCI application on the smartphone. The server manages all application instances and distributes task semantics through HTTP, including operation sequences, parametric operations, semantic vectors, templates, and template embeddings. Other semantics in Table 1 are calculated by the VCI applications during run-time.

The VCI application utilizes third-party Automatic Speech Recognition (ASR) and Text-To-Speech (TTS) services as the input and output channels. It uses the Android accessibility service API to obtain run-time interface layouts and simulate interaction events during task execution. We designed a dialogue GUI containing the dialogue contents and shortcuts for frequent utterances, as shown in Figure 7. We implemented the GUI with a floating window covering the original applications (e.g., Facebook Messenger); hence, the users cannot get any visual feedback from the original apps. However, the generated VCI can still invoke operation system APIs to access the contents of original applications [2, 34], which are utilized during parameter identification (4.3.2). The dialogue GUI is not necessary as the VCI provides voice feedback, and all the buttons have alternative verbal commands. Users can interact with the VCI hands-freely and eyes-freely.

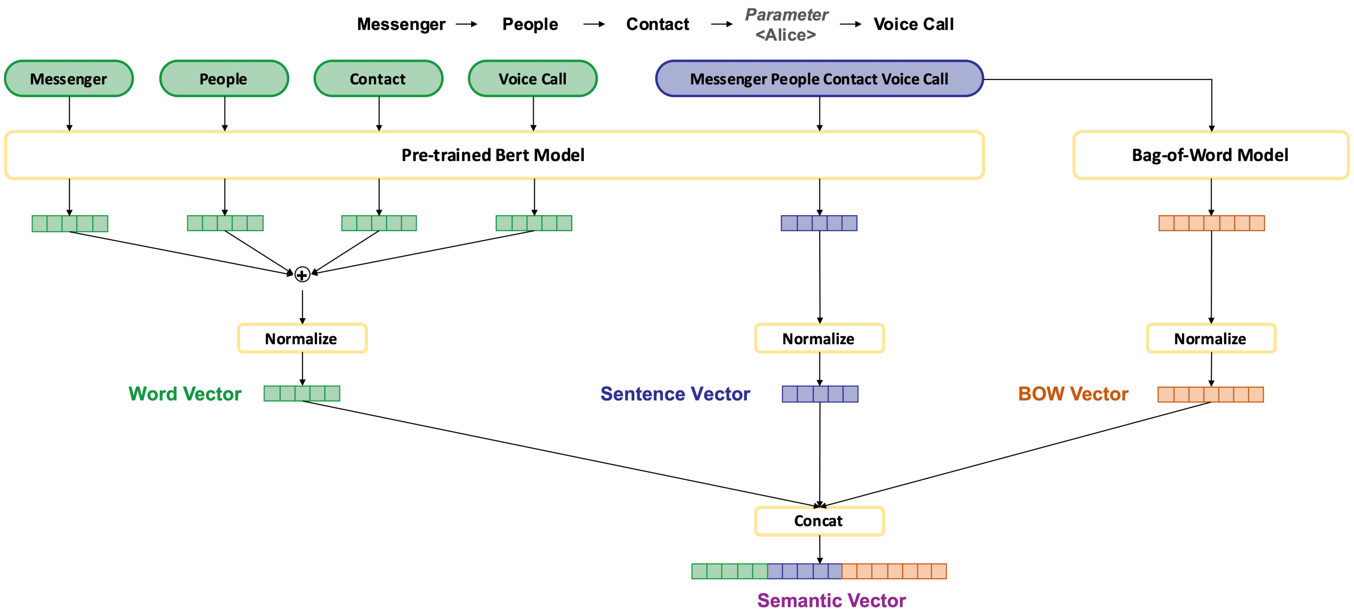
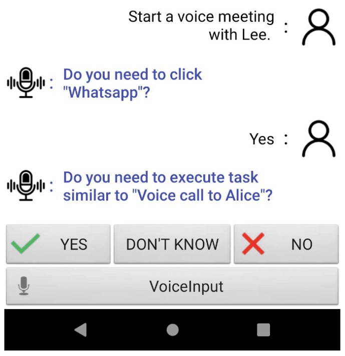


Figure 6: Semantic vector calculation for the task *making a video call with someone*. The detailed operation sequence of the task is shown in Figure 2. The VCI calculates the semantic embeddings of user commands (4.2.2) and templates (4.4.2) in the same way.


Start a voice meeting with Lee. :

: Do you need to click "Whatsapp"?

Yes :

: Do you need to execute task similar to "Voice call to Alice"?

Figure 7: The dialogue GUI, which appears on the devices when users interact with the VCI.

4.2 Intention Recognition

In this section, we will introduce how the generated VCI determines user intention, i.e., the target task that the user wants to carry out after the verbal command.

4.2.1 Template Matching. The VCI tries to match the command with all templates. If any template matches the command successfully, the VCI determines user intention and parameter values and starts task execution. Instead of being provided by the VCI designer, the templates are accumulated from interaction history and are inadequate when the number of VCI users is limited. If none of the templates matches, the VCI calculates and starts a minimized Q&A dialogue.

4.2.2 Similarity Calculation. The VCI calculates the semantic similarities between the command and all supported tasks and constructs a candidate task set. The VCI converts the command into a vector, using the same algorithm and pre-trained model as calculating the task vectors (4.1.2)[2]. The words in the operation sequences are the task descriptions from the application developers, while those in the command are the description from the user. As a result, the command vector is comparable with the task vectors. It then calculates the cosine similarities [61] and takes the zero-mean normalization results as the final similarity scores. The similarity score of the i-th task is calculated as $Similarity\ Score_i = (x_i - \bar{x})/\sigma$. x_i denotes the cosine of the angle between the i-th task and the command. $\bar{x}$ and σ denote the average and standard deviation of all cosines, respectively. As a result, the average and standard deviation of all similarity scores are 0 and 1. Tasks with positive scores are added to the candidate set, as shown in Table 2.

Specially, tasks with scores greater than 2, i.e., exceeding the average by two standard deviations, will skip the *intention filtration* (4.2.3) and jump to the *result confirmation* (4.2.4) directly to simplify the dialogues.

4.2.3 Intention Filtration. The VCI asks multiple yes-no questions based on bag-of-words (BOW) features and distinguishes the target task from the candidate set according to the answers. This approach is similar to the spoken parlor game *20 Questions* [17]. As shown in Table 3, all non-parametric texts in the operation sequences are extracted, each text corresponding to a BOW feature. A task takes the value of one in that feature only if the text appears in its operation sequence.

[2]The command may contain parameter values, which will also be calculated into the command vector. This is the only difference compared with calculating task vectors.

Table 2: An example of candidate set for the command *Start a voice meeting with Lee.* The texts in bold are parameters

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Text in Op1	whatsapp	messenger	whatsapp	messenger	amazon	whatsapp
Text in Op2	chat	people	chat	people	voice input	chat
Text in Op3	**Alice**	contact	**Carol**	contact	-	**Eve**
Text in Op4	**Hello**	**Bob**	voice call	**Chai**	-	contact info
Text in Op5	send	**Hi**	-	voice call	-	clear chat
Text in Op6	-	send	-	-	-	-
Similarity Score	1.510	1.085	1.879	1.794	0.885	0.296

Table 3: BOW Features of tasks in Table 2

BOW Feature	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
whatsapp	1	0	1	0	0	1
messenger	0	1	0	1	0	0
amazon	0	0	0	0	1	0
chat	1	0	1	0	0	1
send	1	1	0	0	0	0
people	0	1	0	1	0	0
contact	0	1	0	1	0	0
voice call	0	0	1	1	0	0
voice input	0	0	0	0	1	0
contact info	0	0	0	0	0	1
clear chat	0	0	0	0	0	1

In each question, the VCI utilizes a BOW feature to ask whether the target task contains a specific operation, such as the purple questions (2 and 4) in Figure 8, and eliminates tasks that are conflict with the answer. We call this kind of questions filtration questions. Since values of all BOW features are zero or one, the user only needs to answer *yes* or *no*, which ensures that no additional semantic understanding failures will be introduced [35, 37]. Users can also reply *I don't know*, in which case the VCI will remove the corresponding BOW feature and restart the Q&A. The user can answer filtration questions based on task semantics and smartphone experiences with a low cognitive burden.

The VCI constructs a weighted decision tree in the BOW space to determine the best order of features with which the number of questions is minimized. It treats the candidate tasks as the samples and the similarity scores as the weights. The decision tree construction algorithm, ID3 [42], segments the samples recursively using features with the greatest information entropy gain and generates a small tree based on a greedy algorithm. Hence the expected depth of the tree, i.e., the number of filtration questions, is minimized. Figure 8 shows an example of the decision tree, where each internal node (in green) corresponds to a feature and each leaf node (in blue) to a candidate task.

The Q&A is a process of interaction-controlled tree traversal with a cursor starting from the root and ending when reaching a leaf. When the cursor comes to an internal node, the VCI asks a filtration question and moves to a child node based on the answer. When it reaches a leaf node, the VCI determines the target task. Tasks with higher similarity scores are closer to the root, meaning a smaller number of filtration questions.

Intention filtration guarantees accurate semantic understanding no matter how difficult and complex the commands are. The mechanism is essential though it is not frequently used since the BERT-based semantic embedding is accurate and the similarity scores of the targets are greater than two in most cases.

4.2.4 *Result Confirmation.*

The VCI asks explicitly whether the user wants to trigger the identified task during the result confirmation. The confirmation question contains a detailed task description. We proposed two mechanisms to describe the task.

If any templates of the same task has already been recorded during semantic accumulation (4.4.1), the VCI describes the task with that template, such as the orange question in Figure 8(6). We choose the most similar template if multiple templates have been accumulated. The similarity is defined as the cosine distance between the command vector and the template embedding. To protect the users' privacy, we fill in the parameters with those used by the designer instead of the previous users. For example, the template is *Video call <>* (Figure 3-A1) and the parameter value used by the designer is *Chai* (Figure 2, operation 3). Hence, the task description is *Video call Chai* (Figure 3-B2).

The VCI describes the operations sequentially (3-A6) if users answer *I don't know* to the confirmation question from the templates, or the VCI has not accumulated any template for the task. For non-parametric operations, the VCI describes the operation types and elements. For operations containing parameters, the VCI

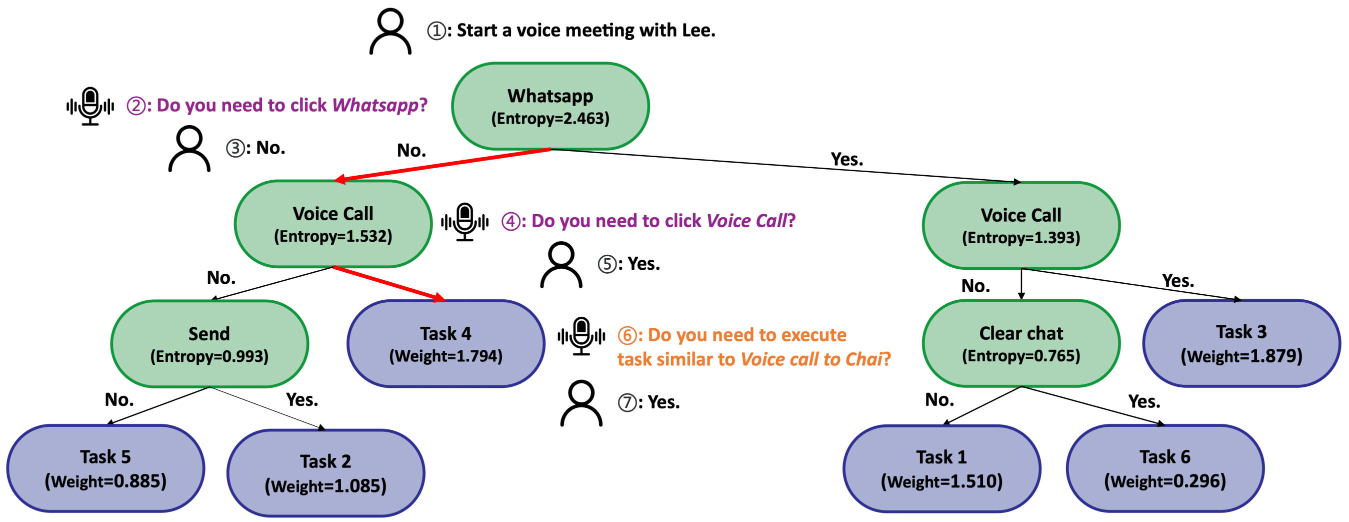

①: Start a voice meeting with Lee.
②: Do you need to click Whatsapp?
③: No.
Whatsapp (Entropy=2.463)
No.
Yes.
Voice Call (Entropy=1.532)
④: Do you need to click Voice Call?
⑤: Yes.
Voice Call (Entropy=1.393)
No.
Yes.
Send (Entropy=0.993)
Task 4 (Weight=1.794)
⑥: Do you need to execute task similar to Voice call to Chai?
⑦: Yes.
Clear chat (Entropy=0.765)
Task 3 (Weight=1.879)
Task 5 (Weight=0.885)
Task 2 (Weight=1.085)
Task 1 (Weight=1.510)
Task 6 (Weight=0.296)

Figure 8: The decision tree constructed from Table 3 and an extra dialogue corresponding to the red arrows. The filtration questions are marked in purple and the confirmation question is marked in orange.

only describes the operation types as the concrete parameter values have not been determined yet.

Intention recognition finishes if the user answers *yes* to the confirmation question, and the next is run-time parameter identification. The described task may not be the target, usually due to inaccurate similarity scores or incorrect answers during the Q&A. The VCI eliminates the rejected task and restarts the intention recognition in this case.

4.3 Task Execution & Parameter Identification

In this section, we mainly discuss how we identify parameter values from the user command. The parameter identification is simultaneous with task execution so that the VCI can fully utilize semantics from run-time GUI.

4.3.1 Task Execution. The VCI simulates touch events step-by-step according to the operation sequence to automate the target task. The critical problem is how to locate the operation elements in the run-time GUI. For non-parametric operations, the VCI calculates the similarities between the element in the operation sequence and all elements in the run-time GUI and chooses the most similar one as the next operation element. The similarity of a run-time GUI element is calculated as follows:

(1) The initial similarity is 0.
(2) If the element shares the same text or content description with that in the operation sequence, its similarity increase by 0.6. Note that most manipulable elements have a text or a description according to accessibility regulations.
(3) We split the smartphone screen into 4*4 blocks. If the element locates in the same block as that in the operation sequence, we add 0.3 to its similarity.
(4) If the aspect ratio difference between the element and that in operation sequence is within 10%, we add 0.2 to its similarity.

For operations with parameters, the VCI identifies parameters from the command to select the correct items from the lists (for *list parameters*) or to determine the texts to input (for *text parameters*).

4.3.2 Parameter Identification. The VCI identifies *list parameters* by GUI parameter searching and *text parameters* by VCI parameter questioning. Unlike approaches based entirely on NLP algorithms, the VCI does not determine the parameter values until they are used during the execution. As a result, we can utilize insights from run-time GUI to realize parameter identification.

GUI Parameter Searching. For *list parameters*, the values are determined by text-matching between the commands and run-time GUI elements, since the GUI contains optional values for the *list parameters*, as shown in Figure 9. Because the sizes of mobile interfaces are always limited, the VCI tries to scroll the list to get more items. The text-matching-based parameter identification makes full use of run-time GUI without any additional user interactions. The GUI parameter searching may fail to match or match multiple candidates, in which case the VCI will ask parameter questions as discussed below.

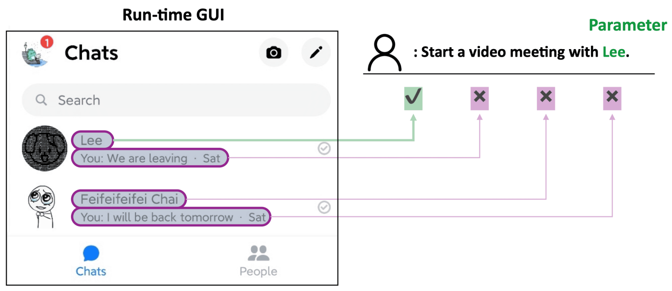

Run-time GUI
Parameter
Chats
Search
Lee
You: We are leaving · Sat
Feifeifeifei Chai
You: I will be back tomorrow · Sat
Chats
People
: Start a video meeting with Lee.
✓
✗
✗
✗

Figure 9: An example of parameter searching. The text *Lee* in *Start a video meeting with Lee* is identified as the parameter.

VCI Parameter Questioning. For *text parameters* and *list parameters* with GUI searching failures, the VCI proposes parameter questions, the answers to which are the parameter values. Parameter values may not be present in the commands, and users can supplement the original commands [45] by answering parameter questions.

The parameter question is composed of information from the last and the next operation, as shown in Figure 10. Given that there are strong semantic relationships between operations and their results, the VCI describes the last operation to assist the user in identifying the current page (the result of the last operation) where the parameter is used. Another essential information is in which list to select or in which edit box to input text. The VCI describes the next element with its label, i.e., the text around or inside it [14]. For instance, in Figure 10, the text box to be edited in run-time GUI has a label *Flying to*. Some GUI elements do not have labels, usually because the absence does not cause confusion and the GUI developers eliminated the labels. In this case, the VCI only describes the type of the next element.

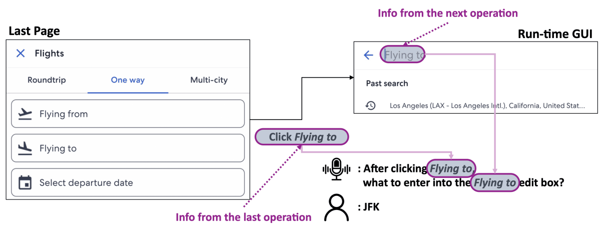

Info from the next operation
Last Page
Run-time GUI
Flights
Roundtrip
One way
Multi-city
Flying from
Flying to
Select departure date
Flying to
Past search
Los Angeles (LAX - Los Angeles Intl.), California, United Stat...
Click *Flying to*
: After clicking *Flying to*, what to enter into the *Flying to* edit box?
: JFK
Info from the last operation

Figure 10: An example parameter question. The answer, *JFK* is identified as the parameter.

4.4 Semantic Accumulation

To reduce extra dialogues for subsequent commands, the VCI generates templates and updates task semantic vectors after executing a command successfully. The semantics are shared among **all VCI users** to accelerate the accumulation. This recurrent strategy gradually improves the semantic understanding ability and simplifies or even avoids extra dialogues.

4.4.1 Template Generation. The VCI transforms the command into a template by eliminating parameters. For example in Figure 11(A), *LAX* are removed from the command; hence the template is *Search a hotel near* <>. The templates are used to match subsequent commands (Figure 11(C-a)) and simplify confirmation questions, which have been discussed in 4.2.1 and 4.2.4, respectively.

4.4.2 Template Embedding. The VCI also calculates the template embedding and updates the semantic vector of the task. The VCI transforms the generated template into a vector with the same strategy in Figure 6. It then adds the semantic vector and the template embedding together, the result of which is regarded as the new semantic vector. The target task can get higher similarity scores based on the updated embedding, which helps simplify or avoid intention filtration, as shown in Figure 11(C-b). Besides, the VCI also proposes better confirmation question leveraging the template embedding, as discussed in 4.2.4.

5 USER STUDY

The user study contained two phases. The first phase evaluated whether the end-users could answer extra questions to automate tasks with the generated VCI. In addition, it assessed AutoVCI's ability to map verbal commands to GUI tasks and the feasibility of semantic accumulation. The evaluations are generally focused on the semantic understanding aspect of AutoVCI, rather than evaluating the VCIs produced by the approach from a user experience perspective. In the meantime, we received few feedback on the user interface, which will be further enhanced in future work as needed. The second phase evaluated the semantic accumulation with a larger scale of data.

5.1 VCI Generation

We used AutoVCI to generate a VCI for the user study. Table 4 shows the applications and tasks supported by the generated VCI. Most of the tasks were not supported by common commercial voice interfaces such as Siri and Google Assistant. Tasks on WeChat were used in the tutorial, while the others were for the formal experiment. It took only 31 minutes to generate the VCI, including 10 minutes to collect operation sequences (around 13.41 seconds per task) and 21 minutes to inspect and correct errors. All 45 tasks contained 206 operations, and 3 of them were incorrectly identified (accuracy of parametric operation identification: 98.5%). All 3 misidentified operations were *entering function names in a search box*, and were manually corrected.

5.2 Phase 1: Evaluating the Semantic Understanding

5.2.1 Procedures. We first introduced the VCI to the participant, and showed an example interaction using a task on WeChat. The participant then tried the other four tasks on WeChat with the assistance of the study moderator. To fully demonstrate the interaction details, we disabled the semantic accumulation module during the tutorials.

In the formal experiment, each participant finished 40 different tasks in random order via the VCI. The tasks were displayed as screenshots of the operation sequences. We did not provide natural language descriptions to avoid users simply parroting the given words, and to increase the variety of user commands. The screenshots disappeared after users activated the VCI. We also provided several parameter candidates to avoid failures due to illegal values. Those values varied for every participant to prevent any side-effects to semantic understanding. Users could interrupt the voice feedback and give answers in advance. When encountering any problem preventing the task from continuing, the participant moved on to the next task. The VCI accumulated semantics automatically after every successful task. The experiment ended with a questionnaire to collect subjective feedback. The participants sequentially completed the study, and the VCI accumulated semantics from each experiment.

We strictly restricted the retries of tasks. Users might give wrong commands or responses, which would be attributed to carelessness if the users discovered the errors on their own before semantic understanding failures happened. In this case, users could retry the task. Otherwise, the answers to the confirmation questions would

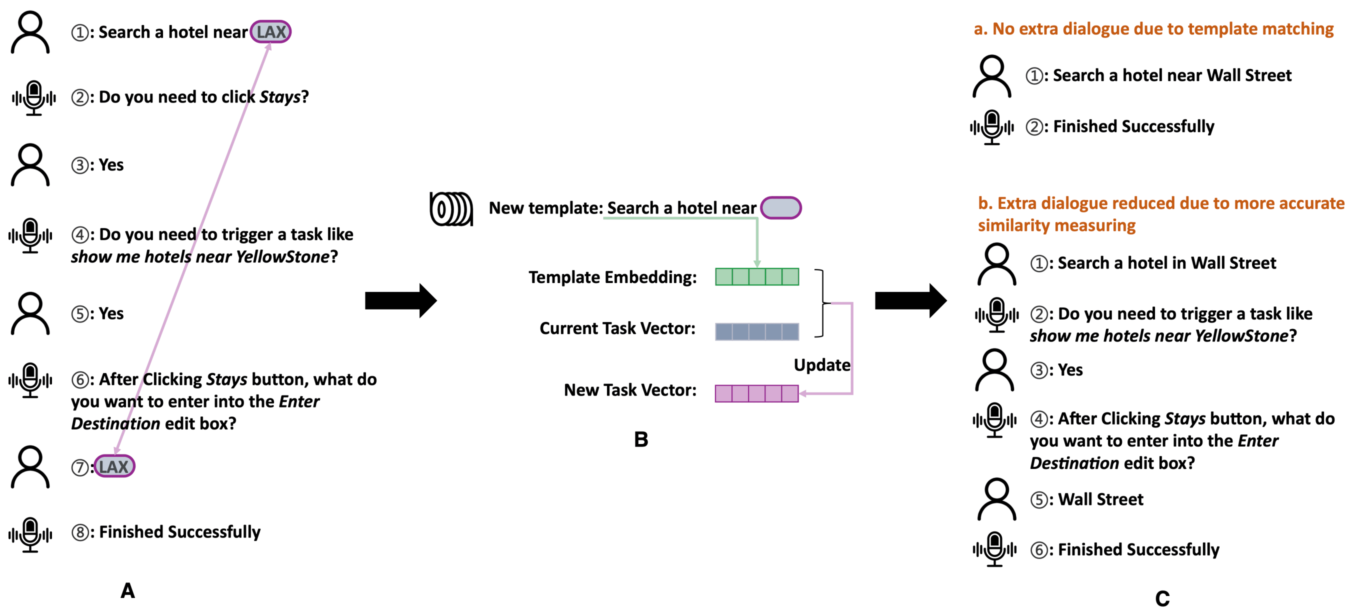

①: Search a hotel near LAX
②: Do you need to click *Stays*?
③: Yes
④: Do you need to trigger a task like *show me hotels near YellowStone*?
⑤: Yes
⑥: After Clicking *Stays* button, what do you want to enter into the *Enter Destination* edit box?
⑦: LAX
⑧: Finished Successfully
A
New template: Search a hotel near
Template Embedding:
Current Task Vector:
Update
New Task Vector:
B
a. No extra dialogue due to template matching
①: Search a hotel near Wall Street
②: Finished Successfully
b. Extra dialogue reduced due to more accurate similarity measuring
①: Search a hotel in Wall Street
②: Do you need to trigger a task like *show me hotels near YellowStone*?
③: Yes
④: After Clicking *Stays* button, what do you want to enter into the *Enter Destination* edit box?
⑤: Wall Street
⑥: Finished Successfully
C

Figure 11: An example of semantic accumulation. A: the dialogue during cold-start; B: accumulated semantics, including a new command template and an update to the task semantic vector; C: the accumulation simplifies subsequent interactions. It also simplifies confirmation questions, which has been discussed in 4.2.4.

Table 4: An overview of the applications and tasks supported in the user study

Application	Category	# Tasks	# Steps per Task	# *list parameters* per Task	# *text parameters* per Task	# Tasks Supported by Siri/Google Assistant
WeChat	Message	5	4.80 (sd=1.17)	0.80 (sd=0.4)	0.40 (sd=0.49)	1/0
DingTalk	Message	5	5.60 (sd=1.74)	1.60 (sd=1.36)	0.40 (sd=0.49)	0/0
Taobao	Shopping	5	5.00 (sd=0.89)	0.60 (sd=0.49)	0.40 (sd=0.49)	0/0
Alipay	Pay	3	6.00 (sd=0.82)	1.00 (sd=0.82)	0.33 (sd=0.47)	0/0
MeiTuan	Delivery	3	5.33 (sd=0.47)	0.00 (sd=0.00)	1.00 (sd=0.00)	0/0
NetEase Music	Music	3	4.33 (sd=0.94)	0.67 (sd=0.47)	0.33 (sd=0.47)	0/0
Amap	Map	3	4.00 (sd=2.16)	0.00 (sd=0.00)	0.67 (sd=0.47)	1/0
Wemeet	Meeting	4	4.00 (sd=1.22)	0.50 (sd=0.50)	0.25 (sd=0.43)	0/0
Keep	Fitness	5	4.00 (sd=1.26)	0.20 (sd=0.40)	0.60 (sd=0.49)	1/0
System Tools	Tools	5	2.4 (sd=1.02)	0.00 (sd=0.00)	1.00 (sd=0.89)	3/2
Settings	System	4	5.50 (sd=0.87)	1.00 (sd=0.71)	0.00 (sd=0.00)	0/0
All	**11**	**45**	**4.58 (sd=1.58)**	**0.60 (sd=0.80)**	**0.49 (sd=0.58)**	**6/2**

be *No*, which was regarded as the result of VCI defects, and users should abandon the task.

5.2.2 Participants. We recruited 16 users (12 males and 4 females, aged 20-32). They were all familiar with the applications used in the study, and had experiences of using smartphone voice interfaces.

5.2.3 Results. We will demonstrate our results on the following six aspects.

Success Rate. 630 (98.4%) out of the total 640 commands were successfully understood and executed. As shown in Figure 12, the success rate was 95% for the first user, and reached 100% for the eleventh user and all remaining users. A Mann-Kendall Test [26] showed an upward trend ($p < 0.005$), which should be attributed to the increased semantic understanding ability rather than the increase in user proficiency (learning effects), as all participants were new to the VCI.

Reasons for Errors. Four failures were caused by users giving incorrect answers, probably due to oversights or failures to answer the questions. This failure decreased with semantic accumulation, as users needed to answer fewer questions. Intention filtration failures led to 5 (0.78%) failed commands, where participants answered all questions correctly, but the VCI could not filter out the correct tasks. The VCI did not add the appropriate tasks to the candidate set due to the weak semantic understanding ability during the cold-start. The remaining failure was due to misunderstanding. The third participant misunderstood the task *starting a fitness course* as *searching a fitness course*, and the given command happened to

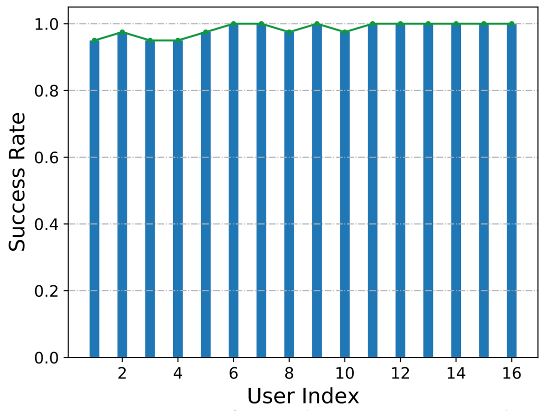


Figure 12: Success rate for each participant. As the user index increased, the VCI accumulated more semantics and improved its command understanding ability.

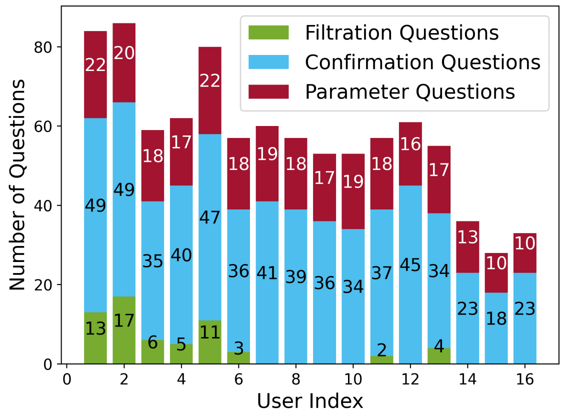
match a template of *searching for takeaways* from another application.

Overall Statistics of Extra Dialogues. Extra dialogues are composed of three types of questions: filtration questions (4.2.3), confirmation questions (4.2.4) and parameter questions (4.3.2). The vast majority were confirmation questions (63.6%), while filtration questions were only a small proportion (6.6%). In terms of questions to understand a single command, 17.8% of the commands did not require any questions. 81.8% of the commands required confirmation questions, nearly half of which also required parameter questions. Only 2.2% of commands required filtration questions.

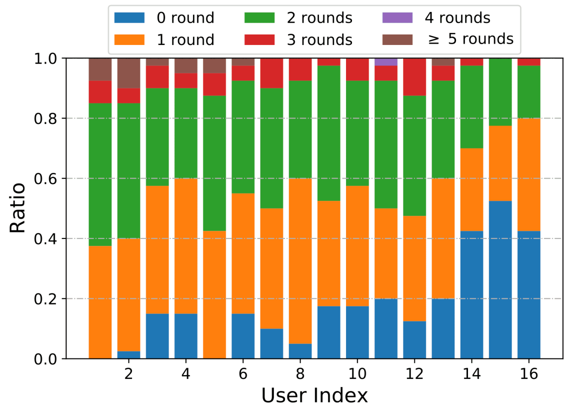
The Decrease in Extra Dialogues. Semantic accumulation can reduce dialogue rounds, i.e., the number of extra questions. Figure 13(a) demonstrates the number of extra questions for each participant. There was a decrease in all three kinds of questions. The average number of questions per command dropped from 2.1 for the first user to a minimum of 0.7 (the fifteenth user). Figure 13(b) shows the distribution of extra dialogue rounds. For the last participant, more than 40% of commands were executed directly, and 80% of commands required at most one-round extra dialogues.

Time Efficiency. As the extra dialogue rounds were reduced, a decrease also occurred in the average interaction time, as shown in Figure 14. We spent an average of 13.14 seconds collecting operation sequences for each task, indicating that the generated VCI was always more efficient than touch-based interaction.

Subjective Feedback. The results of the 7-point Likert scale are displayed in Table 5. In addition to positive feedback, participants also commented that the VCI was promising since it could help accomplish many useful tasks that the existing VCIs do not support. Even though some users mentioned the VCI was more complicated than the existing ones, they felt the complexity was acceptable. Table 7 (in the Appendix) shows the detailed scores from each participant, and there were no clear trends among different users. Subjective ratings were strongly influenced by users' personal preferences and could not be compared among individuals. For example, User 11 and User 13 did not like voice interaction and therefore were reluctant to use the generated VCI. Future work can compare the subjective ratings from a group to better reveal the trend.



(a)



(b)

Figure 13: (a) Numbers of three types of questions answered by different users. (b) Distribution of different extra dialogue rounds for different people.

Table 5: Subjective feedback.

Statements	Results
You can understand the questions easily	6.06 (sd=1.14)
You can answer the questions easily	6.13 (sd=1.05)
You can trigger the functions quickly	6.13 (sd=1.11)
The generated VCI has high intelligence	5.25 (sd=0.75)
The experience is good	5.56 (sd=0.70)
The complicatedness of the VCI is acceptable	5.56 (sd=1.17)
You are willing to use our VCI	6.00 (sd=1.62)

We further discussed with participants why they could answer questions easily (6.13 in 7, as shown in Table 5). The confirmation questions that made up the vast majority were easy to answer because there were obvious semantic similarities in the instructions for the same task (*Start a voice meeting with Lee* vs. *Voice call to*

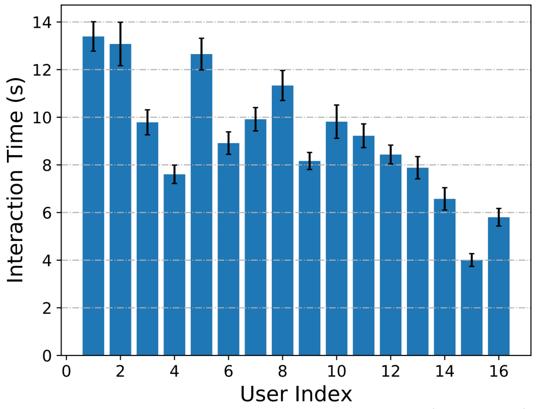


Figure 14: Average interaction time pre task. Error bar indicates one standard error.

Chai in Figure 8). If the VCI described operations sequentially (Figure 3-A), users could judge based on keywords in the descriptions rather than memorizing the entire operation sequence in advance. Parameter questions were also easy to answer. Users could figure out the parameters of a given task according to its inherent semantics without any hints from GUI or VCI. They only judged which parameter was being asked. An important clue was the type of parameters since our VCI almost only asked questions for *text parameters*. Filtration questions were the least frequent. Some users (P3, P5) did have an accurate memory of the application contents. However, most of the participants gave responses depending on the task semantics. Taking the task *starting a video call* as an example, when asked whether to click *Video Call*, users might be confused between *Video Call* and *Video Chat*. However, they still answered *yes* based on the semantics.

5.3 Phase 2: A Larger-Scale Online Test

The goal of phase 2 was to evaluate the semantic accumulation ability with more users and data. The VCI only learned from 16 participants in the first phase. To increase the number of times a single task being triggered and accelerate semantic accumulation, we randomly selected eight tasks for this phase, as shown in Table 6.

5.3.1 Procedure. We constructed a VCI supporting all tasks in Table 4. The participants watched a tutorial video first and finished the eight tasks online in random order. All the interaction details were the same as those in phase 1 except the task execution, which was not relevant to semantic accumulation and was simulated in the browser. The experiment for each participant lasted about 10 minutes.

5.3.2 Results. We recruited 67 users, who generated a total of 536 commands and corresponding dialogues. There was significant degradation in the quality of the data collected online. We deleted 24 (4.48%) failed tasks, mainly because the commands were irrelevant to the given tasks. Two primary metrics measuring semantic understanding ability were proposed:

- Ratio of template match: the percentage of commands which was matched by templates;
- Rounds of extra dialogues per command: the average number of questions to understand a command.

The results proved that the command understanding ability was continuously improving with a larger scale of semantic accumulation. We calculated the average metrics in a window of 64. The window slid through the collected chronological command list to better reveal the trend, as shown in Figure 15(a). Figure 15(b) showed the results in each window. About 75% of commands could be matched by templates in the last window, and each command only introduced a 0.4-round extra dialogue on average. The former metric improved by 5.7 times while the latter decreased by 71.7% compared with those in the first window.

6 OFFLINE EVALUATIONS

We simulated the process of the VCI understanding user commands and evaluated critical metrics to reveal the performances of the different parts of the VCI. We conducted the offline evaluation using the 640 commands with target task annotations collected from the first phase of the user study.

6.1 Command Understanding Ability without any Semantic Accumulation

6.1.1 Simulation Procedure. The VCI ran as normal during the simulation, but we deactivated the semantic accumulation module. We assumed that users could answer all extra questions correctly. Parameter identification depended entirely on text matching or additional parameter questions and could not reflect the semantic understanding ability. Therefore, it was not involved in the simulation.

We input all 640 instructions into the VCI and calculated the following metrics to measure its understanding ability: 1) top-n accuracy, which refers to the percentage of commands whose target task is within the top-n most similar tasks; 2) similarity scores of target tasks; 3) depth of target tasks in the weighted decision trees.

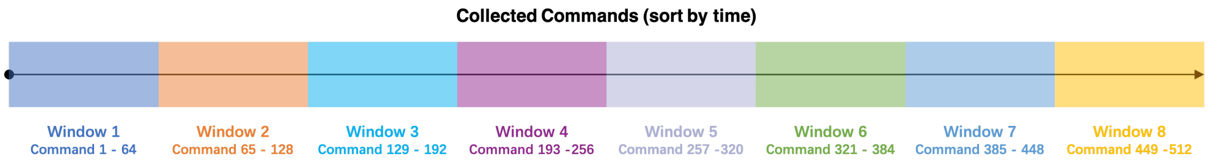
6.1.2 Simulation Results. We will introduce our results in the following two aspects.

Similarities between commands and tasks. The precision and recall of top-1 similarity were 0.801 and 0.772. Figure 16(a) shows the confusion matrix. The command intention classification worked well in most cases, indicating that the algorithm in Figure 6 can effectively model the semantics of both the commands and the operation sequences . However, the accuracy of very few VCI tasks (Task 7, 33, 39) was quite low, indicating that extra dialogues were necessary for accurate semantic understanding during cold-start. Figure 16(b) show how accuracy increased as n increased.

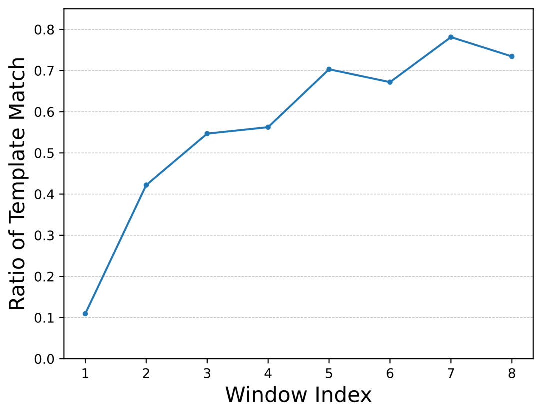
The target tasks of most commands (90.63%) had similarity scores greater than or equal to 2, as shown in Figure 16(c). The VCI could determine intentions by confirmation questions for those commands. Multi-round confirmations were necessary if the target tasks were not the most similar ones. The percentage of commands whose target tasks' similarity scores were less than 0 was only 1.88%. The VCI could not understand these commands without semantic

Table 6: Information about tasks in Phase 2.

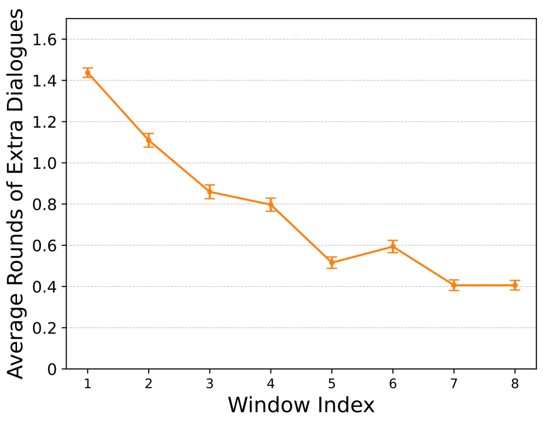
Index	Description	# Operations	# *List Parameters*	# *Text Parameters*	# Triggered Successfully
1	Visit Online Store	5	1	0	58
2	Rename Chatting Group	5	1	1	64
3	Join Online Meeting	3	0	1	67
4	Start Countdown	2	1	0	66
5	Take a Picture	2	0	0	67
6	Add Cellphone Credit	6	2	0	65
7	Clear Shopping Cart	6	0	0	67
8	Join Fans Club	5	0	1	58
	SUM	**34**	**5**	**3**	**512**



(a)



(b)



(c)

Figure 15: (a) We calculated metrics in the sliding window to reveal the trend; (b) The ratio of template match increased as window index increased; (c) The rounds of extra dialogues decreased as window index increased. The error bar indicates one standard error.

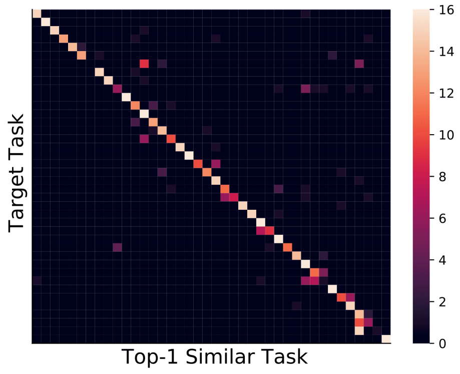
accumulation. Other commands (orange in Figure 16(c)) would go through filtration based on weighted decision trees.

Intention filtration based on weighted decision tree. We introduced two baselines: 1) the depth in the unweighted tree and 2) the similarity rank, simulating VCI proposing confirmation questions sequentially. The VCI finished intention filtration with the least extra dialogues via weighted decision tree, as shown in Figure 17. The decision trees could significantly reduce dialogue rounds compared with ranks. Weighted decision trees outperformed unweighted ones, indicating that the similarity scores were accurate even if the absolute values were insignificant (≤ 2).

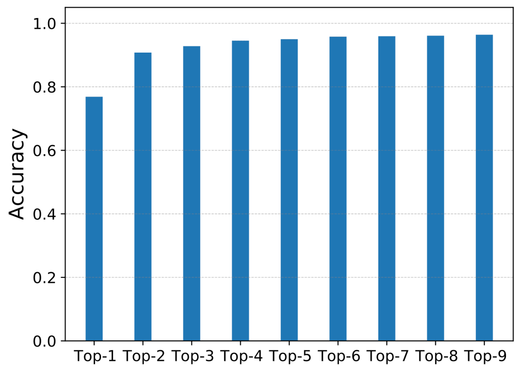
6.2 Improvement of Command Understanding Ability with Semantic Accumulation

6.2.1 Simulation Procedure. The simulated semantic accumulation was the same as that in the actual running times. A simulation session contained 16 epochs corresponding to the 16 participants in the user study. Each epoch was composed of 40 commands of different target tasks. The VCI processed those commands sequentially and accumulated semantics after every command. It accumulated more data with the increase of the epoch index.

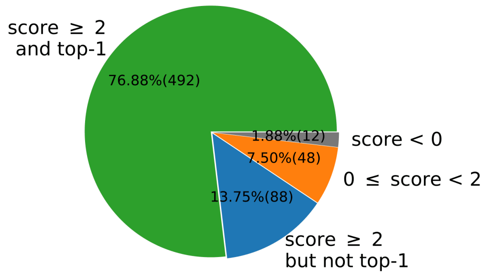
We calculated how the following metrics changed among different epochs to prove the improvement of semantic understanding ability: 1) ratio of template match; 2) top-1 similarity accuracy; 3) the number of commands that went through filtration; 4) the



(a)



(b)



(c)

Figure 16: (a) Confusion matrix of top-1 similarity; (b) Top-*n* similarity accuracy; (c) Similarity scores of target tasks.

number of different kinds of questions. We did not count parameter questions as these questions were reduced due to template match, which was measured by the first metric.

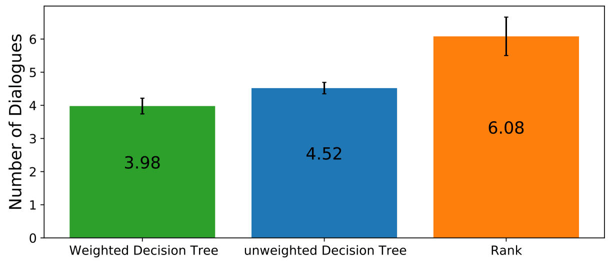


Figure 17: Number of extra dialogue rounds for three different intention filtration approaches. The error bar indicates one standard error.

We shuffled the commands 100 times and calculated the average results for the same epoch to avoid random noise, as the command order affects semantic accumulation. As shown in Figure 18, we sorted the commands by user index and task index. We first shuffled the commands with the same task index and then with the same epoch index. The commands were fed into the VCI according to the permutation.

6.2.2 Simulation Results.

Success Rate. The success rates for the first and second epoch were 98.50% and 99.95%, respectively. The success rate was always 100% after the second epoch. These results also proved that the generated VCI could understand any reasonable commands with a little accumulation.

Different parts of the generated VCI. Figure 20 indicates how different metrics changed with semantic accumulation. The ratio of template match (1) was improving steadily, which reached more than 40% in the last epoch. Top-1 similarity accuracy (2) converged to 1 quickly, which proved the effectiveness of the function vector update strategy. Semantic accumulation also simplified extra interactions. The numbers of filtration questions (4), confirmation questions (5) and total questions (6) decreased by 98.4%, 53.4% and 64.7% from the first epoch to the last one, respectively. The average number of confirmation questions and total questions per extra conversation [3] also decreased. The average number of questions during the filtration process was fluctuating, mainly due to the small number of commands that went through filtration (3).

Different accumulation strategies. We also compared different semantic accumulation strategies while other VCI modules remained the same. Compared with only updating semantic representation or only accumulating templates, our VCI achieved a better performance combining these two methods, as shown in Figure 19. The combination brought a further 43.6% reduction comparing with the two separated methods in the last epoch. This remarkable out-performance was because semantic vectors and command templates were complementary in nature. Templates could reduce extra dialogues to 0, but not those of unmatched instructions. In contrast, updates of vector representations could reduce the extra dialogue of all commands but only reduce to at least one round.

[3] not per command, since many commands did not require extra conversations.

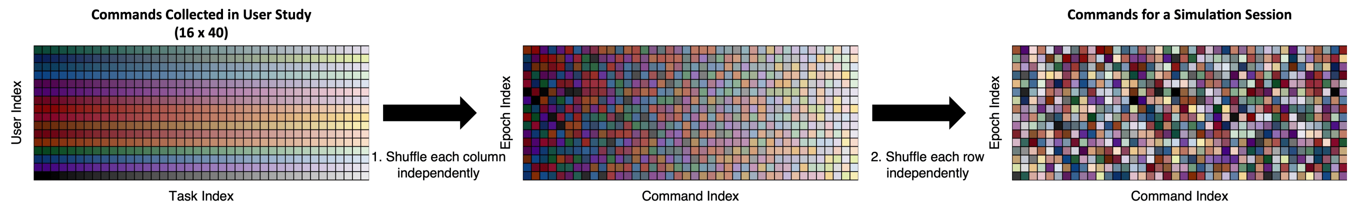

Commands Collected in User Study (16 x 40)
User Index
Task Index
1. Shuffle each column independently
Epoch Index
Command Index
2. Shuffle each row independently
Commands for a Simulation Session
Epoch Index
Command Index

Figure 18: We shuffled the commands to avoid the effect of random noise. Each block represents a command. This process repeated 100 times during the simulation.

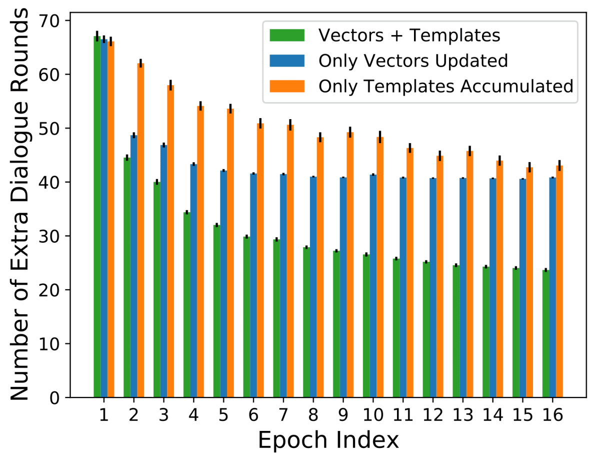


Figure 19: Performances of different semantic accumulation strategies. Error bar indicates one standard error.

7 DISCUSSION

7.1 Contribution to the Field of Voice Interaction

Voice interaction is a hot topic in the human-computer interaction community, and previous research has identified many challenges, such as speech recognition [55, 64], interaction naturalness [64], non-verbal features [64], interaction cognitive burden [46, 55], discoverability [43], etc. AutoVCI mainly addresses the challenge of command semantic understanding [55, 64]. We propose a self-learning strategy that guarantees accurate comprehension of user commands without gathering training corpus for the command recognizer or programming. The voice interface combines task semantics and a small number of extra dialogues to ensure correct semantic understanding, and learns from dialogue contents to improve the understanding ability of subsequent commands. This strategy also significantly improves the learnability [19, 24] of the voice interface, and users are free to use the commands they are used to instead of memorizing command patterns or parameter lists. Our strategy can be applied to generate arbitrary task-oriented VUIs, or can be integrated into existing VUIs such as database accessing [49], programming [37], and data visualization [22], as a backup solution for unmanageable commands.

Another contribution is the mechanism to generate task-oriented VCIs for GUI tasks automatically. AutoVCI makes full use of the task semantics encoded in the GUI and extra dialogues to enable VCI generation without any corpus and programming. Although implemented in the Android platform, AutoVCI can be directly migrated to other GUI platforms because all GUIs share a similar design paradigm, and AutoVCI can extract task semantics in the same way. AutoVCI can be used on non-GUI platforms (e.g., smart speakers). The VCI designers only need to address the two problems caused by lacking texts in non-GUI platforms: 1) how to compute the initial task semantic vector, and 2) how to formulate easy-to-understand filtration and confirmation questions. This automatic generation mechanism can be useful for various stakeholders. The application developer can integrate a VCI quickly into their software, while the voice assistant designer can deploy a VCI supporting many third-party applications. End-users can also create voice shortcuts for frequent tasks. The generated VCIs are suitable in all existing VCIs' scenarios as they only require voice input and feedback.

7.2 Why not a Designer Study

We did not conduct a study to evaluate whether the designers could construct VCIs easily and quickly with AutoVCI because the optimization is obvious compared with traditional approaches. The designers no longer collect corpus, label semantics, or train models. Previous work [38] has already proved that the operation sequences can be collected quickly with little effort.

7.3 Privacy Protection

Some participants expressed their concerns about privacy. The details of the conversations are not uploaded or shared among users. Only semantic vectors and templates are stored in the cloud, where the users do not have direct access to. Even if someone accessed those data, they have no way to infer the original texts according to the vectors. The instruction templates do not contain any parameters, and the remaining parts are prepositions or verbs, which do not contain any sensitive data. When describing tasks with previous instructions, we fill in the templates with the parameters used by the designer instead of using the original commands from earlier users. On the designer side, they can replace private content during VCI generation to ensure their information security.

8 LIMITATIONS & FUTURE WORK

Convergence rate. We did not explore the underlying factors that influence the speed of the semantic accumulation in this paper. Researchers can focus on the relationship among user habits, the

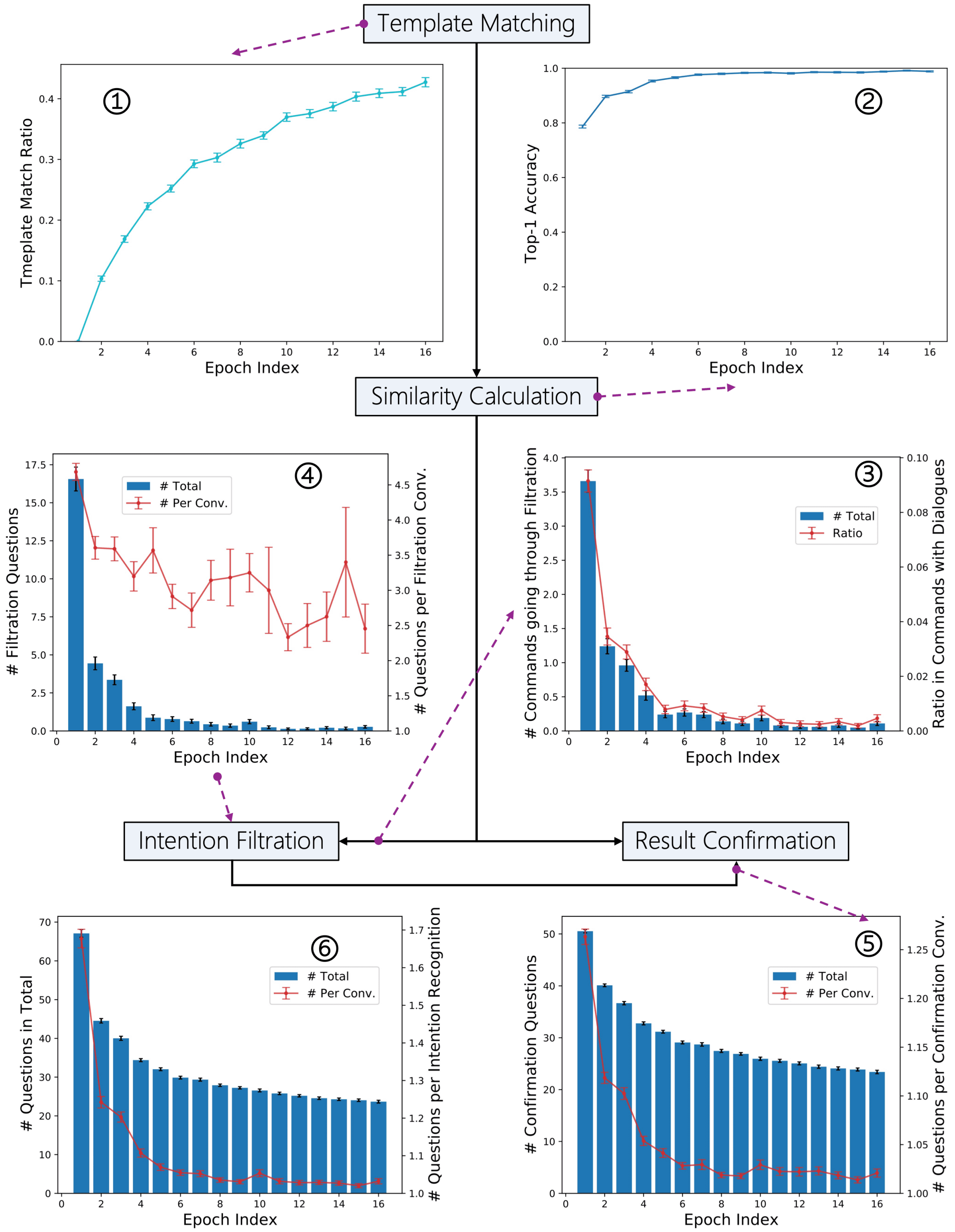


Figure 20: Different metrics (average results of 100 sessions) changed with semantic accumulation. The error bars indicate one standard error. The dashed arrows indicate the figures' positions in the flow. 1) Ratio of template match; 2) Top-1 accuracy; 3) Number of commands going through filtration. The red line indicates the ratio in all commands not matched by template; 4) Number of filtration questions. The red line indicates the average number of questions in a filtration process; 5) Number of confirmation questions. The red line indicates the average number of confirmation questions in an extra conversation; 6) Number of questions. The red line indicates the average number of questions in an extra conversation.

complexity of the operation sequences, different languages, and semantic learning curves in the future.

Reducing extra interactions. We can further reduce the number of extra questions while maintaining a high success rate. Confirmation questions and parameter questions took up the vast majority. Some confirmations were unnecessary after the VCI accumulating enough semantics. The decrease of parameter questions was due to templates, and no accumulated semantics were utilized if none of the templates matched. Future works can integrate more NLP algorithms to reduce those questions.

Evaluating user experience. We did not focus on the user experience in this study, especially for non-lab scenarios. Outside the laboratory, user tolerance for extra dialogues may decrease, but they tend to use similar instructions [13], which can enhance the feasibility of semantic accumulation. Future work can focus more on user experience in different scenarios.

In addition, AutoVCI may fail to map user commands to GUI tasks in some cases. We propose possible solutions which the future work can apply to solve the problems.

Execution failures caused by application updates. The application version updates may change the operation sequences, resulting in execution failures. Both VCI designers and end-users can provide new operation sequences after the updates. The remote server can maintain the mapping from the application versions to the sequences. As a result, the new operation sequences need to be provided only once and the application updates can be seamless to all users. The task semantics and possible commands are not affected by application updates, so the accumulated vectors and templates are still usable for semantic understanding. The semantics encoded in the new operation sequences can also be accumulated to the task semantic vectors.

Incorrect answers from users. Currently, incorrect answers inevitably lead to inaccurate semantic understanding results. The possible solutions can be divided into two aspects. On the one hand, AutoVCI can integrate existing neural network techniques [38] to generate questions that are more natural and easier to understand. On the other hand, AutoVCI can apply a probability model to process user answers. In this case, a wrong answer will only reduce the probability of a correct result rather than directly reject it.

Filtration failures due to little semantic accumulation. As shown in Figure 16(c), 1.88% of instructions could not be correctly understood without semantic accumulation because their similarity scores were less than 0, and AutoVCI did not add them to the candidate set. One possible solution is to extend the candidate set to all tasks. However, this solution will cause users to answer many filtration questions. Users may prefer to terminate the conversations early and modify their instructions.

Same templates for different tasks. Different tasks with the same semantics may have the same templates. For example, both Facebook Messenger and Whatsapp can support *Send a video call to Alice*. AutoVCI may search the application name in the commands or propose extra questions for the target applications because it is unlikely two tasks with the same semantics will appear in the same application. In addition, AutoVCI can maintain the relationship between parameter values and applications. For example, if AutoVCI has recorded that *Alice* only appears in the contact of Facebook Messenger, *Send a video call to Alice* will no longer be ambiguous.

9 CONCLUSION

In this paper, we proposed AutoVCI, an automatic voice command interface (VCI) generation and improvement approach that facilitates designers to construct VCIs with smartphone operation sequences. The generated VCI commands leverages the run-time GUI and the hybrid semantics from the operation sequences to understand and execute verbal commands. In the situation of ambiguity, it launches an extra Q&A dialogue to facilitate accurate command understanding, and learns from the dialogue data to improve its semantic understanding ability. The user study ($N = 16$) showed that the generated VCI worked well and was easy to use with a success rate of 98.3%. In addition, a larger scale online phase ($N = 67$) further validated that the interaction burden of the user decreased as the VCI accumulated semantics - after the accumulation and learning, more than 70% of commands could be understood directly, and the complementary dialogues were only 0.4 rounds on average. Moreover, the offline evaluation demonstrated the feasibility of the semantic understanding strategy that combined NLP algorithms and interaction semantics encoded both in GUI and extra dialogues. This strategy can be adapted to existing and future natural language interaction systems. We hope this work can effectively bootstrap new VCIs on smartphones and improve the semantic understanding ability of a variety of voice user interfaces.

ACKNOWLEDGMENTS

This work is supported by National Key R&D Program of China under Grant No. 2020AAA0105200, Beijing Academy of Artificial Intelligence (BAAI), and the Natural Science Foundation of China under Grant No. 62132010. Our work is also supported by Beijing Key Lab of Networked Multimedia, the Institute for Guo Qiang, Tsinghua University, Institute for Artificial Intelligence, Tsinghua University (THUAI). We would like to thank Zhilin Zhang for his great help in addressing presentation issues during the revise and resubmit phase.

REFERENCES

[1] James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. Plow: A collaborative task learning agent. In *AAAI*, Vol. 7. 1514–1519.
[2] Android. 2021. AccessibilityService | Android Developers. Retrieved August 27, 2021 from https://developer.android.com/reference/android/accessibilityservice/AccessibilityService
[3] Android. 2021. Make apps more accessible | Android Developers. Retrieved August 27, 2021 from https://developer.android.com/guide/topics/ui/accessibility/apps#describe-ui-element
[4] Android. 2021. MediaProjection | Android Developers. Retrieved August 27, 2021 from https://developer.android.com/reference/android/media/projection/MediaProjection
[5] Android. 2021. MotionEvent | Android Developers. Retrieved August 27, 2021 from https://developer.android.com/reference/android/view/MotionEvent
[6] Android. 2021. Support different languages and cultures | Android Developers. Retrieved August 27, 2021 from https://developer.android.com/training/basics/supporting-devices/languages#CreateDirs
[7] V. Antila, J. Polet, A. Lämsä, and J. Liikka. 2012. RoutineMaker: Towards end-user automation of daily routines using smartphones. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*. 399–402. https://doi.org/10.1109/PerComW.2012.6197519
[8] Vikas Ashok, Yevgen Borodin, Yury Puzis, and I. V. Ramakrishnan. 2015. Capti-Speak: A Speech-Enabled Web Screen Reader. In *Proceedings of the 12th International Web for All Conference* (Florence, Italy) *(W4A '15)*. Association for Computing Machinery, New York, NY, USA, Article 22, 10 pages. https://doi.org/10.1145/2745555.2746660

[9] Vikas Ashok, Yury Puzis, Yevgen Borodin, and I.V. Ramakrishnan. 2017. Web Screen Reading Automation Assistance Using Semantic Abstraction. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces* (Limassol, Cyprus) *(IUI '17)*. Association for Computing Machinery, New York, NY, USA, 407–418. https://doi.org/10.1145/3025171.3025229

[10] Amos Azaria, Jayant Krishnamurthy, and Tom M. Mitchell. 2016. Instructable Intelligent Personal Agent. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, 2681–2689.

[11] Marcos Baez, Florian Daniel, and Fabio Casati. 2019. Conversational web interaction: proposal of a dialog-based natural language interaction paradigm for the web. In *International Workshop on Chatbot Research and Design*. Springer, 94–110.

[12] Erin Beneteau, Olivia K. Richards, Mingrui Zhang, Julie A. Kientz, Jason Yip, and Alexis Hiniker. 2019. Communication Breakdowns Between Families and Alexa. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300473

[13] Frank Bentley, Chris Luvogt, Max Silverman, Rushani Wirasinghe, Brooke White, and Danielle Lottridge. 2018. Understanding the Long-Term Use of Smart Speaker Assistants. 2, 3, Article 91 (Sept. 2018), 24 pages. https://doi.org/10.1145/3264901

[14] Syed Masum Billah, Vikas Ashok, Donald E. Porter, and I.V. Ramakrishnan. 2018. SteeringWheel: A Locality-Preserving Magnification Interface for Low Vision Web Browsing. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173594

[15] Dan Bohus and Alexander I. Rudnicky. 2005. Error Handling in the RavenClaw Dialog Management Framework. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing* (Vancouver, British Columbia, Canada) *(HLT '05)*. Association for Computational Linguistics, USA, 225–232. https://doi.org/10.3115/1220575.1220604

[16] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[17] Yihong Chen, Bei Chen, Xuguang Duan, Jian-Guang Lou, Yue Wang, Wenwu Zhu, and Yong Cao. 2018. Learning-to-ask: Knowledge acquisition via 20 questions. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1216–1225.

[18] Pietro Chittò, Marcos Baez, Florian Daniel, and Boualem Benatallah. 2020. Automatic generation of chatbots for conversational web browsing. In *International Conference on Conceptual Modeling*. Springer, 239–249.

[19] Eric Corbett and Astrid Weber. 2016. 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*. 72–82.

[20] Allen Cypher and Daniel Conrad Halbert. 1993. *Watch what I do: programming by demonstration*. MIT press.

[21] 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).

[22] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G Karahalios. 2015. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 489–500.

[23] Melinda T. Gervasio, Michael D. Moffitt, Martha E. Pollack, Joseph M. Taylor, and Tomas E. Uribe. 2005. Active Preference Learning for Personalized Calendar Scheduling Assistance. In *Proceedings of the 10th International Conference on Intelligent User Interfaces* (San Diego, California, USA) *(IUI '05)*. Association for Computing Machinery, New York, NY, USA, 90–97. https://doi.org/10.1145/1040830.1040857

[24] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. 2009. A survey of software learnability: metrics, methodologies and guidelines. In *Proceedings of the sigchi conference on human factors in computing systems*. 649–658.

[25] Jonathan Grudin and Richard Jacques. 2019. Chatbots, Humbots, and the Quest for Artificial General Intelligence. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3290605.3300439

[26] Robert M Hirsch, James R Slack, and Richard A Smith. 1982. Techniques of trend analysis for monthly water quality data. *Water resources research* 18, 1 (1982), 107–121.

[27] Ting-Hao Kenneth Huang, Amos Azaria, and Jeffrey P Bigham. 2016. Instructablecrowd: Creating if-then rules via conversations with the crowd. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. 1555–1562.

[28] Mohit Jain, Pratyush Kumar, Ramachandra Kota, and Shwetak N Patel. 2018. Evaluating and informing the design of chatbots. In *Proceedings of the 2018 Designing Interactive Systems Conference*. 895–906.

[29] Jiepu Jiang, Wei Jeng, and Daqing He. 2013. How Do Users Respond to Voice Input Errors? Lexical and Phonetic Query Reformulation in Voice Search. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Dublin, Ireland) *(SIGIR '13)*. Association for Computing Machinery, New York, NY, USA, 143–152. https://doi.org/10.1145/2484028.2484092

[30] Jihyun Kim, Meuel Jeong, and Seul Chan Lee. 2019. "Why Did This Voice Agent Not Understand Me?": Error Recovery Strategy for in-Vehicle Voice User Interface. In *Proceedings of the 11th International Conference on Automotive User Interfaces and Interactive Vehicular Applications: Adjunct Proceedings* (Utrecht, Netherlands) *(AutomotiveUI '19)*. Association for Computing Machinery, New York, NY, USA, 146–150. https://doi.org/10.1145/3349263.3351513

[31] Yea-Seul Kim, Mira Dontcheva, Eytan Adar, and Jessica Hullman. 2019. Vocal Shortcuts for Creative Experts. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300562

[32] Alfred Krzywicki, Wayne Wobcke, and Anna Wong. 2010. An adaptive calendar assistant using pattern mining for user preference modelling. In *Proceedings of the 15th international conference on Intelligent user interfaces*. 71–80.

[33] Tessa Lau, Julian Cerruti, Guillermo Manzato, Mateo Bengualid, Jeffrey P. Bigham, and Jeffrey Nichols. 2010. A Conversational Interface to Web Automation. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology* (New York, New York, USA) *(UIST '10)*. Association for Computing Machinery, New York, NY, USA, 229–238. https://doi.org/10.1145/1866029.1866067

[34] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6038–6049. https://doi.org/10.1145/3025453.3025483

[35] Toby Jia-Jun Li, Jingya Chen, Haijun Xia, Tom M Mitchell, and Brad A Myers. 2020. Multi-Modal Repairs of Conversational Breakdowns in Task-Oriented Dialogs. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 1094–1107.

[36] Toby Jia-Jun Li, Igor Labutov, Xiaohan Nancy Li, Xiaoyi Zhang, Wenze Shi, Wanling Ding, Tom M Mitchell, and Brad A Myers. 2018. APPINITE: A Multi-Modal Interface for Specifying Data Descriptions in Programming by Demonstration Using Natural Language Instructions. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 105–114.

[37] Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M. Mitchell, and Brad A. Myers. 2019. PUMICE: A Multi-Modal Agent That Learns Concepts and Conditionals from Natural Language and Demonstrations. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 577–589. https://doi.org/10.1145/3332165.3347899

[38] Toby Jia-Jun Li and Oriana Riva. 2018. Kite: Building Conversational Bots from Mobile Apps *(MobiSys '18)*. Association for Computing Machinery, New York, NY, USA, 96–109. https://doi.org/10.1145/3210240.3210339

[39] H. Lieberman. 2001. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann Publishers. https://books.google.com.sv/books?id=wM2JYafw11gC

[40] Ewa Luger and Abigail Sellen. 2016. "Like Having a Really Bad PA": The Gulf between User Expectation and Experience of Conversational Agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 5286–5297. https://doi.org/10.1145/2858036.2858288

[41] David Massimo, Mehdi Elahi, and Francesco Ricci. 2017. Learning User Preferences by Observing User-Items Interactions in an IoT Augmented Space. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization* (Bratislava, Slovakia) *(UMAP '17)*. Association for Computing Machinery, New York, NY, USA, 35–40. https://doi.org/10.1145/3099023.3099070

[42] Tom M Mitchell et al. 1997. Machine learning. (1997).

[43] Christine Murad, Cosmin Munteanu, Benjamin R Cowan, and Leigh Clark. 2019. Revolution or evolution? Speech interaction and HCI design guidelines. *IEEE Pervasive Computing* 18, 2 (2019), 33–45.

[44] B. A. Myers. 1986. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, Massachusetts, USA) *(CHI '86)*. Association for Computing Machinery, New York, NY, USA, 59–66. https://doi.org/10.1145/22627.22349

[45] Chelsea M Myers, Anushay Furqan, and Jichen Zhu. 2019. The impact of user characteristics and preferences on performance with an unfamiliar voice user interface. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–9.

[46] Jun Okamoto, Tomoyuki Kato, and Makoto Shozakai. 2009. Usability Study of VUI consistent with GUI Focusing on Age-Groups. In *Tenth Annual Conference of the International Speech Communication Association*.

[47] Aasish Pappu and Alexander Rudnicky. 2014. Knowledge acquisition strategies for goal-oriented dialog systems. In *Proceedings of the 15th annual meeting of the*

Special Interest Group on Discourse and Dialogue (SIGDIAL). 194–198.

[48] Hannah R.M. Pelikan and Mathias Broth. 2016. Why That Nao? How Humans Adapt to a Conventional Humanoid Robot in Taking Turns-at-Talk. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 4921–4932. https://doi.org/10.1145/2858036.2858478

[49] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*. 149–157.

[50] Chen Qu, Liu Yang, Minghui Qiu, W Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. 2019. BERT with history answer embedding for conversational question answering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1133–1136.

[51] Arpit Rana and Derek Bridge. 2020. Navigation-by-Preference: A New Conversational Recommender with Preference-Based Feedback. In *Proceedings of the 25th International Conference on Intelligent User Interfaces* (Cagliari, Italy) *(IUI '20)*. Association for Computing Machinery, New York, NY, USA, 155–165. https://doi.org/10.1145/3377325.3377496

[52] Lenin Ravindranath, Arvind Thiagarajan, Hari Balakrishnan, and Samuel Madden. 2012. <i>Code in the Air</i>: Simplifying Sensing and Coordination Tasks on Smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems Applications* (San Diego, California) *(HotMobile '12)*. Association for Computing Machinery, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/2162081.2162087

[53] Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-time open-domain question answering with dense-sparse phrase index. *arXiv preprint arXiv:1906.05807* (2019).

[54] Alborz Rezazadeh Sereshkeh, Gary Leung, Krish Perumal, Caleb Phillips, Minfan Zhang, Afsaneh Fazly, and Iqbal Mohomed. 2020. VASTA: A Vision and Language-Assisted Smartphone Task Automation System. In *Proceedings of the 25th International Conference on Intelligent User Interfaces* (Cagliari, Italy) *(IUI '20)*. Association for Computing Machinery, New York, NY, USA, 22–32. https://doi.org/10.1145/3377325.3377515

[55] Jahanzeb Sherwani, Dong Yu, Tim Paek, Mary Czerwinski, Yun-Cheng Ju, and Alex Acero. 2007. Voicepedia: Towards speech-based access to unstructured information. In *Eighth Annual Conference of the International Speech Communication Association*.

[56] Atsushi Sugiura and Yoshiyuki Koseki. 1996. Simplifying Macro Definition in Programming by Demonstration. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology* (Seattle, Washington, USA) *(UIST '96)*. Association for Computing Machinery, New York, NY, USA, 173–182. https://doi.org/10.1145/237091.237118

[57] Ahmad Bisher Tarakji, Jian Xu, Juan A. Colmenares, and Iqbal Mohomed. 2018. Voice Enabling Mobile Applications with UIVoice. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking* (Munich, Germany) *(EdgeSys'18)*. Association for Computing Machinery, New York, NY, USA, 49–54. https://doi.org/10.1145/3213344.3213353

[58] Jesse Thomason, Shiqi Zhang, Raymond Mooney, and Peter Stone. 2015. Learning to Interpret Natural Language Commands through Human-Robot Dialog. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina) *(IJCAI'15)*. AAAI Press, 1923–1929.

[59] Zhen Tu, Yali Fan, Yong Li, Xiang Chen, Li Su, and Depeng Jin. 2019. From Fingerprint to Footprint: Cold-Start Location Recommendation by Learning User Interest from App Data. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 1, Article 26 (March 2019), 22 pages. https://doi.org/10.1145/3314413

[60] Wayne Ward and Sunil Issar. 1994. *Recent improvements in the CMU spoken language understanding system.* Technical Report. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE.

[61] Wikipedia. 2021. Cosine similarity - Wikipedia. Retrieved August 27, 2021 from https://en.wikipedia.org/wiki/Cosine_similarity

[62] Linda Wulf, Markus Garschall, Julia Himmelsbach, and Manfred Tscheligi. 2014. Hands Free - Care Free: Elderly People Taking Advantage of Speech-Only Interaction. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational* (Helsinki, Finland) *(NordiCHI '14)*. Association for Computing Machinery, New York, NY, USA, 203–206. https://doi.org/10.1145/2639189.2639251

[63] Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Yechen Xu, Kai Sun, Dian Yu, Cong Yu, et al. 2020. Clue: A chinese language understanding evaluation benchmark. *arXiv preprint arXiv:2004.05986* (2020).

[64] Nicole Yankelovich, Gina-Anne Levow, and Matt Marx. 1995. Designing SpeechActs: Issues in speech user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 369–376.

[65] Jennifer Zamora. 2017. I'm Sorry, Dave, I'm Afraid I Can't Do That: Chatbot Perception and Expectations. In *Proceedings of the 5th International Conference on Human Agent Interaction* (Bielefeld, Germany) *(HAI '17)*. Association for Computing Machinery, New York, NY, USA, 253–260. https://doi.org/10.1145/3125739.3125766

[66] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. 2010. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics* 1, 1-4 (2010), 43–52.

[67] Ye Zhang and Byron Wallace. 2015. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820* (2015).

[68] Yu Zhong, T. V. Raman, Casey Burkhardt, Fadi Biadsy, and Jeffrey P. Bigham. 2014. JustSpeak: Enabling Universal Voice Control on Android. In *Proceedings of the 11th Web for All Conference* (Seoul, Korea) *(W4A '14)*. Association for Computing Machinery, New York, NY, USA, Article 36, 4 pages. https://doi.org/10.1145/2596695.2596720

A SUBJECTIVE FEEDBACK DETAILS

Table 7: Subjective feedback details.

User Index	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
You can understand the questions easily	3	6	4	7	7	6	6	7	6	7	7	7	5	6	6	7
You can answer the questions easily	7	5	6	5	7	4	7	4	7	7	7	6	6	6	7	7
You can trigger the functions quickly	7	6	6	7	4	6	7	4	7	7	6	7	4	7	6	7
The generated VCI has high intelligence	5	6	5	5	5	5	6	4	6	5	4	6	4	6	6	6
The experience is good	5	5	5	5	6	5	6	7	6	6	5	6	4	6	6	6
The complicatedness of the VCI is acceptable	5	6	5	5	4	5	6	6	6	6	3	7	4	7	7	7
You are willing to use our VCI	6	5	7	6	6	7	7	7	6	7	2	7	2	7	7	7