Multi-Modal Repairs of Conversational Breakdowns in Task-Oriented Dialogs

Toby Jia-Jun Li[1], **Jingya Chen**[1], **Haijun Xia**[2], **Tom M. Mitchell**[1], **Brad A. Myers**[1]
[1]Carnegie Mellon University, [2]UC San Diego
{tobyli, tom.mitchell, bam}@cs.cmu.edu, jingyach@andrew.cmu.edu, haijunxia@ucsd.edu

ABSTRACT
A major problem in task-oriented conversational agents is the lack of support for the repair of conversational breakdowns. Prior studies have shown that current repair strategies for these kinds of errors are often ineffective due to: (1) the lack of transparency about the state of the system's understanding of the user's utterance; and (2) the system's limited capabilities to understand the user's verbal attempts to repair natural language understanding errors. This paper introduces SOVITE, a new multi-modal (speech plus direct manipulation) interface that helps users discover, identify the causes of, and recover from conversational breakdowns using the resources of existing mobile app GUIs for grounding. SOVITE displays the system's understanding of user intents using GUI screenshots, allows users to refer to third-party apps and their GUI screens in conversations as inputs for intent disambiguation, and enables users to repair breakdowns using direct manipulation on these screenshots. The results from a remote user study with 10 users using SOVITE in 7 scenarios suggested that SOVITE's approach is usable and effective.

Author Keywords
Conversational interfaces; conversational breakdown; chatbots; grounding in communication; breakdown repair; disambiguation; instructable agents; GUI semantics

INTRODUCTION
Conversational user interfaces have become increasingly popular and ubiquitous in our everyday lives, assisting users with tasks from diverse domains. However, despite the advances in their natural language understanding capabilities, prevailing conversational systems are still far from being able to understand the wide range of flexible user utterances and engage in complex dialog flows [22]. These agents employ rigid communication patterns, requiring that users adapt their communication patterns to the needs of the system instead of the other way around [5, 27]. As a result, *conversational breakdowns*, defined as failures of the system to correctly understand the intended meaning of the user's communication, often occur. Conversational breakdowns decrease users' satisfaction, trust, and willingness to continue using a conversational system [6, 16, 25, 26, 44], and may cause users to abandon the current task [3]. In this paper, we study repair strategies for such conversational breakdowns, specifically in the context of spoken task-oriented conversations with a mobile device, grounded in the apps and the display of that mobile device.

pair methods in human-agent conversased only the natural language modality. , which refers to the act of adjusting the commodate the listener [5] in the context of conversational agents, is commonly found in users' breakdown repair attempts. Many repair strategies are used in these adjustments, depending on the user's understanding of the cause of the breakdown [5, 48]. For example, when the user suspects that the system has misheard the utterance, they might apply prosodic changes (adjust the rhythm or cadence of speech), overarticulation (exaggerating sounds), increased volume, or simply repetitions of the original utterance [5]. They might make syntactical adjustments to the original utterance if they thought the system did not understand its syntactic structure [5]. Similarly, they might perform semantic adjustments and modifications, such as replacing a word with its synonym, defining a concept, or breaking down a procedure, if they suspected the incorrect semantic understanding to be the cause of the breakdown [5, 48].

However, these strategies are often ineffective for two reasons: First, users often lack an accurate understanding of the cause of the breakdown because current conversational agents do not provide sufficient transparency into the system's state of understanding [5]. Understanding *why* a breakdown happens is crucial for the user to repair it [3]. In current agents, breakdowns are often discovered by users only after the system has acted incorrectly based on its misunderstanding (i.e., performing the wrong action) or from a generic error response (e.g., "Sorry I don't understand") [5]. Thus, little useful information is available for users to infer the cause of the breakdown. The system will provide a task-specific clarification response only in the small portion of cases where a developer has explicitly programmed an error handling conversation flow for the specific breakdown scenario.

Second, even when users correctly identify the causes of the breakdowns, their repairs in natural language are often ineffective [5]. This is especially problematic in breakdowns

UIST'20, October 20–23, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7514-6/20/10.
DOI: https://doi.org/10.1145/3379337.3415820

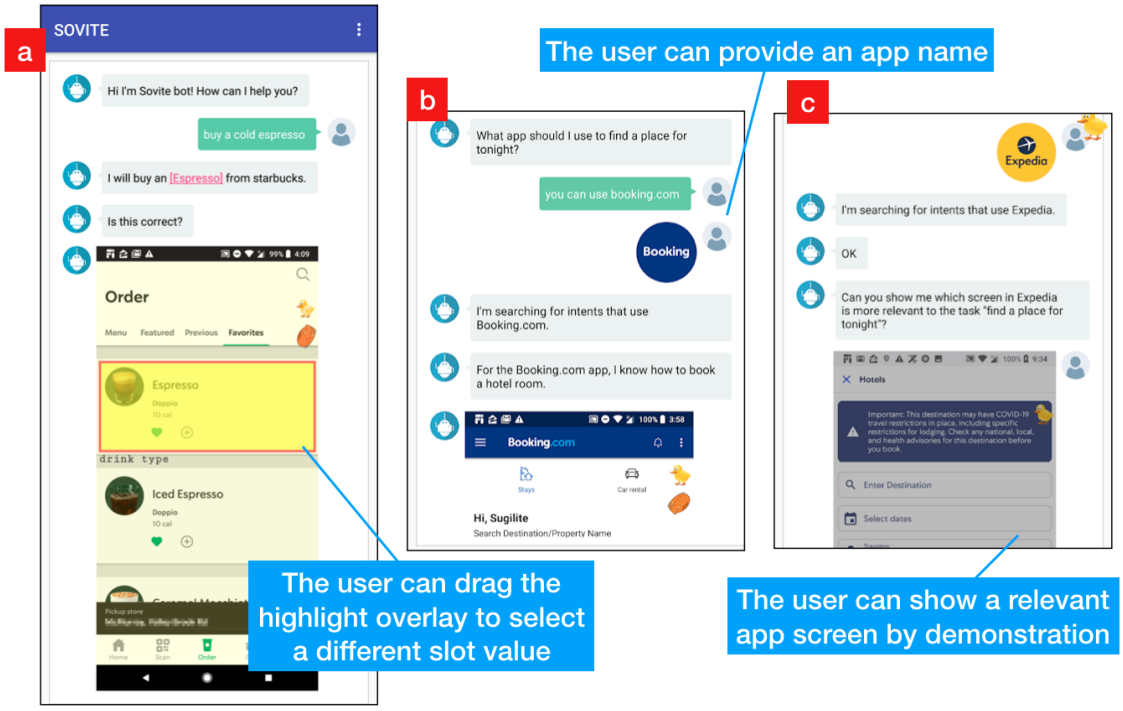

Figure 1. The interface of SOVITE: (a) SOVITE shows a app GUI screenshot to communicates its state of understanding. The yellow highlight overlay specifies the task slot value. The user can drag the overlay to fix slot value errors. (b) To fix intent detection errors, the user can refer to an app that represents their desired task. SOVITE will match the utterance to an app on the phone (with its icon shown), and look for intents that use or are relevant to this app. (c) If the intent is still ambiguous after referring to an app, the user can show a specific app screen relevant to the desired task.

caused by natural language understanding errors, where the user needs to make semantic and syntactic adjustments such as verbally defining a keyword, explaining a procedure, replacing a word with its synonym, or restructuring the sentence instead of simply repeating the utterances with exaggerating sounds. The user usually needs to *guess* the right strategy to use due to the lack of transparency into the system's natural language understanding models. Existing conversational systems also often have problems understanding these repairs due to the system's limited capability of reasoning with natural language instructions and common sense knowledge [39]. This is part of the reason why commercial systems like Siri and Alexa rarely ask users to try to repair the conversation, but just immediately perform generic fallback actions such as searching the web.

Visualizing the agent's understanding of user intent is a promising way to address these challenges. Some existing agents support displaying visual responses (known as *cards*[1] in Google Assistant) within natural language conversations. These cards allow the user to *see* the agent's state, and to interact with the agent visually to complement speech commands. However, these cards must be hard-coded by the developers, require significant effort to create, and therefore have limited adoption across task domains. In this paper, we propose a new approach to address these challenges leveraging the graphical user interfaces (GUIs) of existing apps. This approach can be applied on devices with screens and access to app GUIs, such as smartphones and smart display devices (e.g., Amazon Echo Show, Google Home Hub). Our approach helps the user discover breakdowns and identify their causes by showing the system's state of understanding using GUI screenshots of the underlying apps for the task domain. It subsequently allows the users to repair the breakdowns by using direct manipulation on the screenshots and by making references to relevant apps and screens in the conversation.

The use of app GUIs has many advantages. GUIs and conversational agents are two different types of interfaces to mostly *the same* set of underlying computing services, since most, if not all, supported task domains in task-oriented conversational agents have corresponding mobile apps. The GUIs of these apps encapsulate rich knowledge about the flows of the tasks and the properties and relations of relevant entities [40]. The majority of users are also familiar with the app GUIs, the way to interact with them, and their general design patterns [36], which makes them ideal mediums through which the agent and the user can communicate in a supplementary modality during natural language conversations.

This paper introduces SOVITE[2], a new interface that helps users discover, identify the causes of, and recover from conversational breakdowns in task-oriented agents using this app-grounded multi-modal approach (Figure 1). A remote user study of 10 participants using SOVITE in 7 conversational breakdown scenarios showed that end users were able to successfully use SOVITE to fix common types of breakdowns caused by natural language understanding errors. The participants also found SOVITE easy and natural to use.

[1] https://developers.google.com/assistant/conversational/rich-responses

[2] SOVITE is named after a type of rock. It is also an acronym for **S**ystem for **O**ptimizing **V**oice **I**nterfaces to **T**ackle **E**rrors.

This paper makes the following contributions:

1. A new multi-modal approach that allows users to discover, identify the causes of, and repair conversational breakdowns caused by natural language understanding errors in task-oriented agents using the GUIs of existing mobile apps.
2. The SOVITE system: an implementation of the above approach, along with a user study evaluating its effectiveness and usability.

PROBLEM SETTING

Frame-Based Task-Oriented Conversational Agents

SOVITE is designed for handling natural language understanding errors in task-oriented conversational agents that use the *frame-based* architecture for dialog management [8]. This is the most popular architecture used by the majority of existing commercial task-oriented agents [29]. It is based on a set of *frames* (also known as *intents*). Each frame represents a kind of intent that the system can extract from user utterances. A frame often contains *slots* whose values are needed for fulfilling the underlying task intent. For a user utterance, the system first determines the frame to use (e.g., finding a flight), and then tries to fill the slots (e.g., date, departure airport and arrival airport) through the dialog. Frames can ensure the necessary structures and constraints for task completion [29], which are lacking in statistical dialog approaches (e.g., deep reinforcement learning for response generation [33,34,57] and information retrieval-based structures [66]) often used in social chatbots that converse with human users on open domain topics without explicit task completion goals.

A typical pipeline for a frame-based dialog architecture consists of multiple steps, including: (1) speech recognition, (2) natural language understanding, (3) natural language generation, and (4) speech synthesis [29]. Among those, (2) natural language understanding causes most of the critical breakdowns, as reported in Myers et al.'s study [48] on how users overcome obstacles in voice user interfaces. For the (1) speech recognition step, state-of-art algorithms have reached human parity [67]. For screen-based systems like what we use, the user can easily read and edit the transcription from the speech recognition step. Further, users' natural repair strategies such as repetition, prosodic changes, and overarticulation are effective for speech recognition errors [5]. The (3) natural language generation step in frame-based agents is typically rule-based with manually created templates, therefore less prone to errors compared with statistical approaches commonly used in social chatbots. Finally, the latest techniques for the (4) speech synthesis step reliably produce clear and easy-to-understand synthetic speech from text.

Problem Scope

This work therefore focuses on the breakdowns caused by (2) natural language understanding errors. There are two key components in the natural language understanding step: *intent detection* and *slot value extraction*. Intent detection errors are those where the system misrecognizes the intent in the user's utterance, and subsequently invokes the wrong dialog frame (e.g., responding "what kind of cuisine would you like" for the command "find me a place in Chicago tonight" when the user intends to book a hotel room.) In slot value extraction errors, the system either extracts the wrong parts in the input as slot values (e.g., extracting "Singapore" as the departure city in "Show me Singapore Airlines flights to London.") or links the extracted phrase to incorrect entities (e.g., resolving "apple" in "What's the price of an apple?" to the entity "Apple company" and therefore incorrectly invoking the stock price lookup frame). As illustrated in the examples, slot value extraction errors can cause either the wrong slot match in the dialog frame (the airline example) or the wrong dialog frame (the apple example).

Note that there are other types of breakdowns in task-oriented agents that SOVITE does not handle. It does not address (1) speech recognition errors, as discussed previously. It also does not address breakdowns caused by errors in task fulfillment (i.e., exceptions when executing the task), errors in generating agent response ((3) and (4)), or the user's lack of familiarity with intents (e.g., the user does not know what intents the system can support).

Instructable Agents

An *instructable agent* is a promising new type of frame-based agent that can learn intents for new tasks interactively from the end user's natural language instructions [4, 37, 61] and/or demonstrations [1, 35, 39, 47]. It allows users to use agents for personalized tasks and tasks in "long-tail" domains, addressing the "out-of-domain" errors in human-agent conversations [37].

However, supporting effective breakdown repair is even more challenging for instructable agents. The user-instructed task domains have many fewer example utterances (usually only one) for training the underlying natural language understanding model. As a result, breakdowns caused by natural language understanding errors occur more frequently.

Further, when encountering breakdowns, the instructable agents seldom have task-specific error handling mechanisms for user-instructed task domains. In agents with professionally-developed task domains, developers can explicitly program error handling conversation flows for domain-specific breakdowns [2, 46]. However, end users with little programming expertise seldom create such error handlers for user-taught tasks due to the lack of tool support, and more importantly, the lack of expertise to consider and to handle possible future breakdown situations in advance.

We support instructable agents in SOVITE by (1) supporting user-instructed task domains by not requiring existing domain knowledge, hard-coded error handling mechanisms, or large corpus in the task domains, and (2) supporting the easy transition to user instruction when a breakdown turns out to be caused by out-of domain errors rather than natural language understanding errors.

Design Goals

We have the following design goals:

1. The system should enable users to effectively discover, identify the causes of, and repair conversational breakdowns in intent detection and slot value extraction of frame-based task-oriented conversational agents.

2. The system should handle conversational breakdowns in various task domains, including *user-instructed* ones, without requiring existing domain knowledge or manually-created domain-specific error handling mechanisms.

RELATED WORK

Studies of Breakdowns in Conversational Interfaces

The design of SOVITE is informed by insights from previous studies of breakdowns in conversational agents. As reported in Beneteau et al.'s 2019 deployment study [5] of Alexa, all study participants experienced breakdowns when conversing with the agent. The participants used a variety of repair strategies based on their understandings of the causes of the breakdowns, but those repairs were often not effective since their understandings were frequently inaccurate. Repairing breakdowns caused by natural language understanding errors (compared with speech recognition errors) was particularly problematic, as the natural repair strategies used by users such as semantic adjustments and defining unclear concepts were not well-supported by the agents. Other studies [3, 9, 14, 27, 48, 54] reported similar findings of the types of breakdowns encountered by users and the common repair strategies.

In a 2020 study conducted by Cho et al. [14], more than half of the responses given by Google Home in a user study with five information-seeking tasks were "cannot help" error responses (40%) or "unrelated" responses (24%) that were not useful for the user's request. In most cases, the "cannot help" messages did not provide useful information about the causes of the breakdowns to help the users with breakdown repairs. The participants were sometimes able to infer the causes of the breakdowns from "unrelated" responses, but this process was often unreliable and confusing. The 2018 study by Myers et al. [48] identified "Rely on GUI" as a strategy that users naturally use when they encounter obstacles in conversational interfaces, where they looked at the corresponding app's GUIs to look for cues on what to say for task intents.

Motivated by the these insights, SOVITE helps the users identify the causes of breakdowns by visualizing the system's state of understanding of user intent using the app GUI screenshots.

Assisting Conversational Breakdown Repairing

Some previous approaches have been proposed to assist users with discovering, identifying, and repairing conversational breakdowns. In a taxonomy of conversational breakdown repair strategies by Ashktorab et al. [3], repair strategies can be categorized into dimensions of: (1) whether there is evidence of breakdown (i.e., whether the system makes users aware of the breakdown); (2) whether the system attempts to repair (e.g., provide options of potential intents), and (3) whether assistance is provided for user self-repair (e.g., highlight the keywords that contribute to the intent classifier's decision).

In the results from that paper [3], the most preferred option by the users was to have the system attempt to help with the repair by providing options of potential intents. However, as discussed earlier, this approach requires domain-specific "deep knowledge" about the task and error handling flows manually programmed by the developers [2, 46], and therefore is not practical for user-instructed tasks. In fact, even agents with professionally developed conversational skills such as Alexa and Google Assistant often only provide generic error messages (e.g., "I didn't understand.") with no transparency into the states of understanding in the system and no mechanism for further interactions [14, 54]. In comparison, SOVITE does not require any additional efforts from the developers. It only requires "shallow knowledge" in a domain-general generic language model to map user intents to the corresponding app screens (details in the Implementation section).

The second most preferred strategy in [3] was for the system to provide more transparency into the cause of the breakdown, such as highlighting the keywords that contribute to the intent detection results. This approach is also relevant to work in explainable machine learning (e.g., [62]), which seeks to help users understand the results from intelligent systems and therefore provide more effective inputs. However, these approaches usually require users to verbally clarify or define these keywords. A previous study [39] found that when users tried to verbally explain a concept unknown to the system, they often introduced even more unknown concepts in their explanations. The agents also have problems understanding such explanations due to their limited capability of reasoning with natural language instructions and domain knowledge.

SOVITE further explores the design space of improving system transparency—it visualizes the intent detection and slot value extraction results with app GUI screenshots. It also allows the users to easily repair the breakdowns using references to app GUI contents and direct manipulation on the screenshots.

Multi-Modal Mixed-Initiative Disambiguation Interfaces

SOVITE uses a multi-modal approach [51] that visually displays the system states and enables direct manipulation inputs from users in addition to spoken instructions to repair breakdowns, unlike prior systems that use only the natural language inputs and outputs [2, 3, 46]. SOVITE's design uses the *mutual disambiguation* pattern [50], where inputs from one modality are used to disambiguate inputs for the same concept from a different modality. Similar patterns have been previously used for handling errors in other recognition-based interfaces [45], such as speech recognition [63] and pen-based handwriting [31]. Visually-grounded language instructions were also used in interactive task learning for robots [59] and performing web tasks [1, 56]. SOVITE, to our best knowledge, is the first system to use this pattern to handle natural language understanding errors for task-oriented dialogs with app GUIs.

The design of SOVITE also applies the principles of mixed-initiative interfaces [23]. Specifically, it considers breakdown repairing as a human-agent collaboration process, where the user's goals and inputs come with uncertainty. The system shows guesses of user goals, assists the user to provide more effective inputs, and engages in multi-turn dialogs with the user to resolve any uncertainties and ambiguities. This combination of multi-modal and mixed-initiative approaches has been previously applied in the interactive task learning process in systems such as [1, 30, 36, 39]. SOVITE bridges an important gap in these systems, as they focus on the ambiguities, uncertainties, and vagueness embedded in the user instructions of

new tasks, while SOVITE addresses the conversational breakdowns caused by the natural language understanding problems when users invoke already-supported tasks.

In terms of the technique used, SOVITE extracts the semantics of app GUIs [13, 43] for grounding natural language conversations. Compared with the previous systems that used the semantics of app GUIs for learning new tasks [36, 38, 58], extracting task flows [40], and supporting invoking individual GUI widgets with voice commands [64], a new idea in SOVITE is that it encodes app GUIs into the same vector space as natural language utterances, allowing the system to look up semantically relevant task intents when the user refers to apps and app GUI screens in the dialogues for repairing intent detection errors (details in the Implementation section).

THE DESIGN OF SOVITE

Communicating System State with App GUI Screenshots

The first step for SOVITE in supporting the users in repairing conversational breakdowns is to provide transparency into the state of understanding in the system, allowing the users to discover breakdowns and identify their causes. SOVITE leverages the GUI screenshots of mobile apps for this purpose. As shown in Figure 1a, for the user command, SOVITE displays one or more (when there are multiple slots spanning many screens) screenshots from an app that corresponds to the detected user intent (details of how SOVITE extracts the screenshots and creates the highlight overlays are discussed in the Implementation section). For intents with slots, it shows screens that contain the GUI widgets corresponding to where the slots would be filled if the task was performed manually using the app GUI. SOVITE also adds a highlight overlay, shown in yellow in Figures 1a and 2, on top of the app's GUI, which indicates the current slot value. If the slot represents selecting an item from a menu in the GUI, then the corresponding menu item will be highlighted on the screenshot. For an intent without a slot, SOVITE displays the last GUI screen from the procedure of performing the task manually, which usually shows the result of the task. After displaying the screenshot(s), SOVITE asks the user to confirm if this is indeed the correct understanding of the user's intent by asking, "I will... `[the task]`, is this correct?", to which the user can verbally respond.

For example, as shown in Figure 1a, SOVITE uses the screenshot of the "Order" screen in Starbucks app to represent the detected intent "buy a `[drink type]` from Starbucks", and highlights the value "Espresso" for the slot `drink type` on the screenshot.

Design Rationale

SOVITE's references to app GUIs help with *grounding* in human-agent interactions. In communication theory, the concept of grounding describes conversation as a form of collaborative action to come up with common ground or mutual knowledge [15]. For conversations with computing systems, when the user provides an utterance, the system should provide evidence of understanding so that the user can evaluate the progress toward their goal [10]. As described in the *gulf of evaluation* and *gulf of execution* framework [24,49] and shown in prior studies of conversational agents [3, 5], execution and evaluation are interdependent—in order to choose an effective strategy for repairing a conversational breakdown, the user needs to first know the current state of understanding in the system and be able to understand the cause of the breakdown.

The app GUI screenshots can be ideal mediums for communicating the state of understanding of the system. They show users the *evidence* of grounding [14] through their familiar app GUIs. This approach highlights that the agent performs tasks on the user's behalf, showing the key steps of navigating app screens, selecting menu items, and entering text through GUIs based on slot values as if a human agent was to perform the task using the underlying app. We believe this approach should help users to more effectively identify the understanding errors because it provides better *closeness of mapping* [21] to how the user would naturally approach this task. Prior studies showed that references to existing app GUIs were effective in other aspects of conversational interface designs, such as enabling users to explain unknown concepts [39] and helping users come up with the language to use to invoke app functionalities [48]. Showing the screenshots of GUIs is also a useful way to present the context of a piece of information, making the content easier to understand [42].

Intent Detection Repair with App GUI References

When an intent detection result is incorrect, as evidenced by the wrong app or the wrong functionality of app shown in a confirmation screenshot, or when the agent fails to detect an intent from the user's initial utterance at all (i.e., the system responds "I don't understand the command."), the user can fix the error by indicating the correct apps and app screens for their desired task.

References to Apps

After the user says that the detected intent is incorrect after seeing the app GUI screenshots, or when the system fails to detect an intent, SOVITE asks the user "What app should I use to perform... `[the task]`?", for which the user can say the name of an app for the intended task (shown in Figure 1b). SOVITE looks up the collection of all supported task intents for not only the intents that *use* this underlying app, but also intents that are semantically *related* to the supplied app (we will discuss how SOVITE finds semantically relevant task intents in the Implementation section).

For example, suppose the agent displays the screenshots of the OpenTable app for the utterance "find a place for tonight" because it classifies the intent as "book restaurant" when the user's actual intent is "book hotel". The user notices that the app is wrong from the screenshot and responds "No" when the agent asks, "Is this correct?" The agent then asks which app to use instead, to which the user answers "Booking.com", which is an app for booking hotel rooms. If there was indeed a supported "book a hotel room" intent using the Booking.com app, SOVITE would respond "OK, I know how to book a hotel room using Booking.com", and then show the screenshots of booking hotel rooms in Booking.com for confirmation. If no such intent was available, but there was an intent semantically related to the Booking.com app, such as a 'book a hotel room" intent using the Hilton app, SOVITE would respond "I don't know how to find a place for tonight in Booking.com, but I

know how to book a hotel room using Hilton" and show the corresponding screenshots. The user can verbally confirm performing the task using the Hilton app instead, or indicate that they still wish to use the Booking.com app and teach the agent how to perform this task using Booking.com by demonstration (if the underlying agent supports the user instruction of new tasks; details discussed below).

References to App Screens

In certain situations, the user's intent can still be ambiguous after the user indicates the name of an app; there can be multiple intents associated with the app (for example, if the user specifies "Expedia" which can be used for booking flights, cruises, or rental cars), or there can be no supported task intent in the user-provided app and no intent that meets the threshold of being sufficiently "related" to the user-provided app. In these situations, SOVITE will ask the user a follow-up question "Can you show me which screen in... `[the app]` is most relevant to... `[the task]`?" (shown in Figure 1c). SOVITE then launches the app and asks the user to navigate to the target screen in the app. (The user may also say "no" and start over.) After the user reaches the target screen, they can click on a floating SOVITE icon to provide this screen as an input to SOVITE. SOVITE then finds intents that are the most semantically related to this app screen among the ambiguous ones (technical details in the Implementation section), or asks the user to teach it a new one by demonstration.

Ease of Transition to Out-of-Domain Task Instructions

An important advantage of SOVITE's intent disambiguation approach is that it supports the easy transition to the user *instruction* of a new task if the intent disambiguation attempt fails when the user's intended task is out of scope (i.e., there is no existing intent that supports the task). An effective approach to support handling out-of-scope errors is programming-by-demonstration (PBD) [37]. The state-of-the-art PBD systems (e.g., [35, 39, 58]) can learn new tasks from the user's demonstrations on third-party app GUIs. SOVITE's approach can directly connect to the user instruction mode in these systems. Since at this point in the overall process, SOVITE already knows the most relevant app and app screen for the user's intended task and how to navigate to this screen in the app, it can simply ask the user "Can you teach me how to... `[the task]` using... `[the app]` in this screen", switch back to this screen, and have the user to continue demonstrating the intended task to teach the agent how to fulfill the previously out-of-scope task intent. The user may also start over and demonstrate from scratch if they do not want to start the instruction from this screen.

Design Rationale

The main design rationale of supporting intent detection repairs with app GUI references is to make SOVITE's mechanism of fixing intent detection errors *consistent* with how users discover the errors from SOVITE's display of intent detection results. When users discover the intent detection errors by seeing the wrong apps or the wrong screens displayed in the confirmation screenshots, the most intuitive way for them to fix these errors is to indicate the correct apps and screens that should be used for the intended tasks. Their references to the apps and the screens also allow SOVITE to extract richer semantic context (e.g., the app store descriptions and the text labels found on app GUI screens) than having the user simply rephrase their utterances, helping with finding semantically related task intents (technical details in the Implementation section).

Slot Value Extraction Repair with Direct Manipulation

If the user finds that the intent is correct (i.e., the displayed app and app screen correctly match the user's intended task), but there are errors in the extracted task slot values (i.e., the highlighted textboxes, the values in the highlighted textboxes, or the highlighted menu items on the confirmation screenshots are wrong), the user can fix these errors using direct manipulation on the screenshots.

All the highlight overlays for task slots can be dragged-and-dropped (Figures 1a and 2). For slots represented by GUI menu selections, the user can simply drag the highlight overlay to select a different item. For example, assuming the agent incorrectly selects the item "Espresso" for the utterance "order a cold espresso" due to an error in entity recognition, as shown in Figure 1a, the user can drag the highlight overlay to "Iced Espresso" on the screenshot to specify a different slot value. (If the user's desired slot value is not on the screen, the user can say "no" and indicate the correct screen to use, as discussed in the previous section.) The same interaction technique also works for fixing mismatches in the text-input type slot values. For example, if the agent swaps the order between starting location and destination in a "requesting Uber ride" intent, the user can drag these overlays with location names to move them to the right fields in the app GUI screenshot (Figure 2). When a field is dragged to another field that already has a value, SOVITE performs a *swap* rather than a *replace* so as not to lose any user-supplied data.

Alternatively, when the value for a text-input type slot is incorrect, the user can repair it using the popup dialog shown in Figure 2. After the user clicks on the highlight overlay for a text-input slot, a dialog will pop up, showing the slot's current value in the user's original utterance. The user can adjust the text selection by dragging the highlight boundaries in the identified entities (e.g., the system recognizes the slot value as "The Lego" when the user says "find showtimes for *The Lego Movie*.") The same dialog alternatively allows the user to just enter a new slot value by speech or typing.

Design Rationale

We believe these direct manipulation interactions in SOVITE are intuitive to the users—this is also supported by the results reported in the User Study section. The positions and the contents of the highlight overlays represent where and what slot values would be entered if the task was performed using the GUI of the corresponding app. Therefore, if what SOVITE identified does not match what the users would do for the intended task, the users can directly fix these *inconsistencies* through simple physical actions such as drag-and-drop and text selection gestures, and see immediate feedback on the screenshots, which are major advantages of direct manipulation [60].

Figure 2. SOVITE provides multiple ways to fix text-input slot value errors: *LEFT*: the user can click the corresponding highlight overlay and change its value by adjusting the selection in the original utterance, speaking a new value, or just typing in a new value. *RIGHT*: the user can drag the overlays on the screenshot to move a value to a new slot, or swap the values between two slots.

IMPLEMENTATION

We implemented SOVITE in Java as an Android app. SOVITE was developed and tested on a Google Pixel 2 XL phone running Android 8.0. It does *not* require the root access to the phone, and should run on any phone with Android 6.0 or higher. SOVITE is open-sourced on GitHub[3].

The current implementation of SOVITE builds on our open-sourced SUGILITE system [35]. SUGILITE is an instructable agent that allows end users without significant programming expertise to teach new tasks by demonstrating on the GUIs of existing third-party Android apps.

SUGILITE uses a standard frame-based dialog management architecture with intents, slots, and slot values. Its natural language model uses a SEMPRE [7]-based Floating Parser [53] that can parse the user's utterance into a corresponding expression that invokes an intent and sets the respective slot values. The model was trained on the lexical (e.g., unigrams, bigrams, skip-grams), syntactic (e.g., part-of-speech tags, named-entity tags), and semantic (e.g., word embeddings) features extracted from the training utterances for each task intent, including when users teach new tasks for an utterance (detail in [36, 39]). Because the task fulfillment in SUGILITE is instructed by end users, there are usually only a very small number of sample training utterances (often only one) for each task intent. As a result, conversational breakdowns are common in SUGILITE's interaction with users when they use utterances with diverse vocabulary, structures, or expressions that are not covered in the training corpus.

While we implemented SOVITE with our SUGILITE agent, we believe the approach used in SOVITE should generalize to other frame-based task-oriented conversational agents as well. The only major part of SOVITE's implementation that is specific to SUGILITE is its mechanism for generating app GUI screenshot confirmations. However, there are other practical ways to generate these app GUI screenshot confirmations without relying on the programming by demonstration scripts in SUGILITE (details in next section).

[3]https://github.com/tobyli/Sugilite_development

Generating the App GUI Screenshot Confirmations

In SUGILITE, each supported task intent corresponds to an automation script created from user demonstrations of performing the task manually using the GUIs of the underlying app. Therefore, SOVITE can extract app GUI screenshots for these intents by instrumenting the demonstration process.

When the user starts demonstrating a task, SOVITE creates a virtual display device in the background that mirrors the main display that the user sees for capturing the screenshots. For each GUI action demonstrated by the user, SOVITE takes a screenshot that captures this action. At the end of the demonstration process, SOVITE compares the task slot values with the demonstrated actions to identify actions that correspond to the task slots and saves these screenshots to be used as the confirmation for this demonstration's underlying task intent. For example, assuming the user's demonstration for the task "order an Espresso" contains an action "click on the item 'Espresso'" from a menu on the GUI of Starbucks app, SOVITE will use the screenshot taken from the user demonstrating this action as a confirmation for the intent. The same mechanism also works for slot values from text inputs (e.g., the user demonstrates typing "Chicago" into a textbox for a command "book a hotel room in Chicago").

Although the implementation of generating app GUI screenshot confirmations used in SOVITE, as described above, only applies to programming-by-demonstration instructable agents such as SUGILITE [35], PLOW [1], and VASTA [58], there are other feasible approaches for generating app GUI screenshot confirmations in other types of agents. For example, recent advances in machine learning have been shown to support directly matching natural language commands to specific GUI elements [52] and generating semantic labels for GUI elements from screenshots [13]. For agents that use web API calls to fulfill the task intents, it is also feasible to compare the agent API calls to the API calls made by apps by analyzing the code of the apps (e.g., CHABADA [20]), or to the network traffic collected from the apps (e.g., MobiPurpose [28]). These techniques should allow associating slots with their corresponding app GUI widgets without relying on user demonstrations.

Finding Relevant Intents from Apps and App Screens

When the user refers to an app name or an app screen for their desired task for disambiguating task intents, SOVITE first looks for intents that use exactly this app or this app screen. If none of the supported task intents uses the exact app or app screen that the user refers to, SOVITE can recommend relevant task intents. For example, if the user refers to the app "Booking.com" or the page for "List hotels near `[location]`" in Google Maps to explain the utterance "find a place for tonight", SOVITE can prompt "I know how to book a hotel room using the Hilton app, is this what you want to do?"

The technical challenge here is to identify semantically relevant task intents based on the user-provided app names and app screens. An effective way to match the user's task intent with natural language descriptions of goals (e.g., book hotel) to apps (e.g., Booking.com) is to leverage the app descriptions in the app stores. For example, MessageOnTap [12] uses word embeddings to represent the semantic meanings of individual

words in the user utterances and app store descriptions, and calculates the cosine similarity between the word embedding centroids of each app description and the conversation to recommend relevant apps in human-human conversations (e.g., recommending the Calendar app and the OpenTable app when one party in the conversation says "let's schedule a dinner.") SOVITE uses a similar approach but recommends task intents from the user references to apps instead of recommending apps from the user expressions of task intents as in MessageOnTap.

Specifically, for each app reference made by the user, a remote SOVITE server retrieves its description from the Google Play store, and calculates the sentence embeddings for the app store description and the sample utterances of each supported task intent using a state-of-art pre-trained Sentence-BERT model [55], which is a modified BERT network [19] that can derive semantically meaningful sentence embeddings for calculating semantic relatedness. SOVITE is then able to identify the most relevant task intent for the app by finding the task intent whose centroid of sentence embeddings of all its sample utterances has the highest cosine similarity with the sentence embedding of the target app's app store description. All this can be done in real-time as the user is interacting with SOVITE.

When the user refers to a specific app screen, SOVITE uses a similar technique for finding semantically relevant task intents. The only difference is that instead of using the sentence embeddings of app store descriptions, it uses the embeddings of all the text labels shown on the screen, and computes their semantic relatedness with each supported task intent to find the most relevant one.

USER STUDY

We conducted a remote user study[4] to evaluate SOVITE. The study examined the following two research questions:

RQ1: Can users understand and use SOVITE's new features for identifying and repairing conversational breakdowns?

RQ2: Is SOVITE effective for fixing conversational breakdowns caused by natural language understanding errors in task-oriented agents?

Participants

We recruited 10 participants (2 women, 8 men, ages 25-41) for our study. 5 participants were graduate students in two local universities, and the other 5 worked at different technical, administrative, or managerial jobs. All of our participants were experienced smartphone users with more than 3 years of experience of using smartphones. 8 of the 10 participants (80%) were active users of intelligent conversational agents such as Alexa, Siri, or Google Assistant. Each participant was compensated $15 for their time.

Study Design

The remote study session with each participant lasted 30–40 minutes. After agreeing to the consent form presented online and filling out a demographic survey, each participant received a short tutorial of SOVITE, which showed the SOVITE features discussed above. The participant was then presented with the 7 tasks in random order. In each task, the participant saw an example conversation scenario that contained a user voice command and SOVITE's response (i.e., each scenario tells the participant "Assume you have said `[utterance]`, and here is the agent's response. You need to identify whether the system's understanding of the intent was correct and fix the breakdown using SOVITE when the understanding was incorrect"). The 7 tasks include one "no error" scenario (Scenario 1), and 6 breakdown scenarios that cover different types of intent detection and slot value extraction errors (Table 1). The participant then filled out a post-study questionnaire about their experiences with SOVITE, and ended the study with a short interview with the experimenter.

The study was performed remotely using the Zoom video conference software. SOVITE ran on a Pixel 2 XL phone running Android 8.0 with relevant third party apps pre-installed. We streamed the screen display of the phone through a camera pointing at its screen, so that the remote participant could see and hear the output of the phone. The participant was able to control the phone *indirectly* through the relay of the experimenter: the participant could point to a GUI widget on the phone screencast with the mouse cursor, and ask the experimenter to click on the widget or enter text into the widget on their behalf. The participant could also ask the experimenter to say something by speech to the phone, and the experimenter repeated the participant's utterance exactly. Since speech recognition errors were not a concern for this study, repeating the utterance was not a confound.

Impact of the COVID-19 Pandemic

This study was conducted in April 2020 in the midst of the COVID-19 global pandemic. Due to health concerns, we were unable to conduct an in-person lab study as originally planned. Although in the remote study, the participants were not able to directly control and speak to SOVITE, we believe the results were still valid for the two research questions we asked. Specifically, the study measured whether the users could come up with what to do in SOVITE when presented with breakdown situations, and whether these inputs were effective for breakdown repairs. We tried a few third-party software tools for directly screencasting and remote controlling Android phones but ran into stability and performance issues with them when sharing with remote participants; therefore we used the camera method described above.

Note that this is *not* a Wizard-of-Oz study because the system was actually operating on the participants' utterances and actions—the experimenter just served as an intermediary since we could not have the participants use the actual phone.

Results

Among the 70 scenario instances (10 participants $\times$ 7 scenarios), including 10 "no error" scenarios, the participants correctly identified all 10 "no error" scenarios and discovered 57 out of 60 errors (95%). Among the discovered errors, they successfully fixed *all* of them using SOVITE. The participants failed to notice the error in two instances of Scenario 7 and one instance of Scenario 6. When asked to reflect upon their experience, the participants attributed all of these failure cases

[4]The study protocol was approved by the IRB at our institution.

#	Breakdown Type	Example Scenario	User Repair Method with SOVITE
1	No error	Make a call to Amazon (in the Phone app)	Not applicable
2	Intent: no intent matched (did not understand the command)	Find something to eat (should order food for delivery in Doordash)	Provide a reference to the correct app to use (and the screen if needed)
3	Intent: wrong app used	Find a place for tonight (recognized as using OpenTable for restaurant booking instead of using Hilton for hotel booking	Provide a reference to the correct app to use (and the screen if needed)
4	Intent: correct app, wrong screen used	Book a ticket to New York (recognized as booking a hotel room in Expedia instead of booking a flight)	Provide a reference to the correct screen in Expedia to use
5	Slot: wrong item selected in a menu	Buy an Iced Espresso from Starbucks (the slot value recognized as "Espresso")	Drag the highlight on the screenshot to select the correct item
6	Slot: wrong value extracted for text input	Find the showtimes of The Lego Movie (the slot value recognized as "The Lego")	Click on the highlight on the screenshot to modify the slot value
7	Slot: slot value mismatched	Book an Uber ride to airport from home (the starting location and the destination are swapped)	Drag the highlight on the screenshot to swap slot values

Table 1. The 6 breakdown types and a "no error" type covered in the user study, an example scenario for each type, and their corresponding user repair methods using SOVITE. All participants saw all 7 in random order.

to the "expectation of capabilities" problem, which we will describe in the Discussion section below.

Subjective Results

In a questionnaire after the study, we asked each participant to rate statements about SOVITE's usability and usefulness on a 7-point Likert scale from "strongly disagree" to "strongly agree". SOVITE scored on average 6.1 ($SD = 0.83$) on "*I find* SOVITE *helpful for fixing understanding errors in conversational agents*", 6.4 ($SD = 0.8$) on "*I feel* SOVITE *is easy to use*", and 6.3 ($SD = 0.9$) on "*I'm satisfied with my experience with* SOVITE" Specifically for SOVITE's individual features, the participants rated 6.5 ($SD = 0.81$) on "*The highlights on screenshots for task parameter values are clear*" and 6.2 ($SD = 1.54$) on "*Dragging the highlights to fix parameter errors is natural.*" These results suggest that our participants were positive about the usability and usefulness of SOVITE.

DISCUSSION

We observed some confusion over the highlight overlays in screenshot confirmations in the participants' interactions with SOVITE. A few participants tried to interact with the other GUI components on the screenshots when they first encountered them. For example, in Scenario 4, P1 tried to use the back button on the screenshot to switch to the "book flight" page. In Scenario 7, P2 was looking for a "swap" button on the screenshot. However, after trying to interact with these GUI components and receiving no response, they quickly realized that only the highlight overlays were interactive on the screenshots and successfully repaired the breakdowns thereafter. The overlay is not a common metaphor in interfaces—users are more familiar with static screenshots where nothing is interactive, and actual GUIs where every element can be directly interacted with. The overlay (also known as *interaction proxy* [68]) sits in between, where the user can specify actions *about* an underlying GUI element (e.g., the value that *should go into* a textbox or the menu item that *should be selected*) using direct manipulation, but not directly interact with these GUI elements. A future direction is to explore the design space of overlay interfaces to make them more intuitive to use.

The "expectation of capabilities" problems [3, 5, 48] impacted the user's capability to discover errors in SOVITE in some cases. For example, the visual clue for Scenario 7 was rather subtle on the screenshot (i.e., the highlights for starting location and destination in Uber were swapped, as shown in Figure 2). P7 missed this error and went with "OK" when SOVITE asked for confirmation. When asked about this instance after the study, P7 said "*I didn't expect it [the system] to make this error... I thought the original utterance was clear, so when I saw two highlights saying "home" and "airport" I didn't carefully check the order since I assumed that the system would get it right... I was more looking at if this was indeed the request ride screen in Uber.*" Users may look more carefully at places where they expect errors to appear, but their expectations might not always match the system's behaviors. However, with SOVITE, once the user discovers an error, it is straightforward what the cause of error is and how to fix it, which is a significant improvement from the prevailing systems where the user needs to guess the cause of the error (which is often inaccurate) and come up with the repair strategy to use (which is often ineffective as a result) [5, 48].

Efficiency wise, we did not measure the time-on-task in the study due to the delays and overheads from running the study remotely. But from our observations, while adding some overhead to the conversation, using SOVITE should still be more efficient than completing the tasks manually in most cases. For example, completing the "order coffee" task in Starbucks requires up to 14 clicks on 8 screens. In comparison, the user reads one GUI screenshot confirmation (and fixes the errors, if any) in addition to speaking the initial utterance when using SOVITE for the same task.

Design Implications

SOVITE illustrates the effectiveness of presenting the system's state of understanding in a way closely matched to how the user would otherwise (i.e., not using speech) naturally approach the problem to help with error discovery. While speech is a natural modality for interacting with task-oriented agents, the technical limitations in natural language understanding and reasoning capabilities limit its effectiveness in handling

conversational breakdowns. App GUI screenshots can serve as a good complement to natural language in this context.

An important underlying assumption in SOVITE's strategy of using app GUI screenshots is that users are familiar with the app GUIs. This is also the assumption of many prior interactive task learning systems like [1, 32, 35, 58]. SOVITE requires the user to have a mental model of "apps" so that they understand how to complete their intended tasks through existing app GUIs. Today, this assumption seems reasonable, given the high adoption rate of smartphones [11]. Most app GUIs are designed to be easy-to-use with common design patterns [18], so users are likely able to understand the screens even if they have never used the particular app before. Using app GUIs is still by far the most common means through which the users access computing services. However, it would be interesting to think about how this may change for certain user groups in some task domains in the future. For example, are we going to see the conversational agents become "the native interface" for some tasks and some user groups in the future, just as how GUIs replaced command-line interfaces? In our opinion, the app-GUI-based approach used in SOVITE can be a stepping stone to a more integrated speech-oriented agent in the future, which may eventually transcend the app-oriented design of current smartphones.

SOVITE's design highlights the importance of *consistency* between how users fix the errors and how they discover the errors. Once the users discover the errors (e.g., the wrong app was used, the wrong screen was shown, the highlights are at wrong places, the slot values use the wrong parts of the initial utterances, etc.), the ways to fix them are rather intuitive and obvious (e.g., saying the correct app, pointing to the correct screen, dragging the highlights to the correct places, and selecting the right parts of the initial utterances to be used in the slot values). This was noticed and praised by many participants in our study. With SOVITE, the user no longer needs to guess the strategy to fix the error (e.g., explain a word, replace words with synonyms, restructure the syntax) like with prevailing systems when the error message was the generic "Sorry I didn't understand".

LIMITATIONS AND FUTURE WORK

SOVITE currently does not handle task intents that span multiple apps (even though the underlying SUGILITE system does). Those intents are often higher-level intents that involve multiple smaller sub-intents within individual apps and information exchange between them (e.g., "plan a dinner party" can involve "find time availability for `[people]` in Calendar", "make a restaurant reservation at `[time]`", and "notify the `[people]` about the `[reservation info]`". An interesting future challenge is to design a new confirmation mechanism to clearly show the system's state of understanding for such cross-domain intents with increased complexity.

As discussed previously in the Problem Scope section, SOVITE only handles intent detection and slot value extraction errors in natural language understanding. We hope to integrate SOVITE with the existing mechanisms that handle other kinds of errors, such as voice recognition errors, task execution errors, and feedback generation errors.

SOVITE currently displays the full app GUI screenshots as a part of the conversation. As a result, the screenshots are displayed in about half of their original size, making them harder to read for the users. The highlight overlays for smaller GUI elements are also prone to the "fat finger" problem. To address this issue, one approach is to extend our model for extracting app GUI screenshots so that it only displays parts of the screens that are most relevant to the underlying task intents. This might be feasible as we already have a mechanism to determine the semantic relatedness between GUI screens and task intents, but it risks making it harder for users to understand the context of the displayed portions. Another approach is to add support for zooming and panning using familiar gestures such as pinch-to-zoom on the screenshots.

We plan to explore the challenge of better encoding the semantics of app GUI screens for assisting natural language understanding. SOVITE's current mechanism only takes the text labels shown on GUIs into consideration. In the future, we plan to capture more comprehensive semantics of app GUI screens by leveraging the GUI layouts (e.g., the distance between elements [41] and design patterns [18, 43]), control flows among GUI screens [40], and large collections of user interaction traces. The availability of large-scale GUI datasets like RICO [17] makes future experiments in this area feasible.

Lastly, the user-provided repairs in SOVITE only apply locally to the current dialog session. In the future, we plan to develop mechanisms that allow the conversational agent to *learn* from the user-provided repairs to improve its performance. The user's expression of the actual intent for their natural language command collected through SOVITE can be a highly valuable resource for applying *online learning* (a machine learning approach that supports incremental learning using small batches of data) to improve the accuracy of the agent's natural language understanding models on the fly.

CONCLUSION

Conversational breakdown repairing in task-oriented dialogues is surprisingly little studied or handled by research or commercial intelligent agents. The lack of effective and robust breakdown repair mechanisms significantly affects the adoption of these agents. SOVITE shows an app-grounded multi-modal approach that can effectively help users discover, identify the causes of, and repair breakdowns caused by intent detection errors and slot value extraction errors in certain contexts. We look forward to future collaborations between HCI, behavioral science, and AI/NLP researchers to address this issue in human-agent interactions for all kinds of errors in all contexts.

ACKNOWLEDGMENTS

This research was supported in part by Verizon through the InMind project, a J.P. Morgan Faculty Research Award, NSF grant IIS-1814472, and AFOSR grant FA95501710218. Any opinions, findings or recommendations expressed here are those of the authors and do not necessarily reflect views of the sponsors. We would like to thank our study participants, our anonymous reviewers, and Michael Xieyang Liu, Haojian Jin, and Forough Arabshahi for their helpful feedback.

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 *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2 (AAAI'07)*. AAAI Press, Vancouver, British Columbia, Canada, 1514–1519.

[2] Amazon. 2020. Alexa Design Guide. (2020). https://developer.amazon.com/en-US/docs/alexa/alexa-design/get-started.html

[3] Zahra Ashktorab, Mohit Jain, Q Vera Liao, and Justin D Weisz. 2019. Resilient Chatbots: Repair Strategy Preferences for Conversational Breakdowns. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 254.

[4] Amos Azaria, Jayant Krishnamurthy, and Tom M. Mitchell. 2016. Instructable Intelligent Personal Agent. In *Proc. The 30th AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 4.

[5] 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 (CHI '19)*. ACM, New York, NY, USA, Article 243, 13 pages. DOI: http://dx.doi.org/10.1145/3290605.3300473

[6] Frank Bentley, Chris Luvogt, Max Silverman, Rushani Wirasinghe, Brooke White, and Danielle Lottridge. 2018. Understanding the Long-Term Use of Smart Speaker Assistants. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3, Article Article 91 (Sept. 2018), 24 pages. DOI:http://dx.doi.org/10.1145/3264901

[7] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 1533–1544.

[8] Daniel G Bobrow, Ronald M Kaplan, Martin Kay, Donald A Norman, Henry Thompson, and Terry Winograd. 1977. GUS, a frame-driven dialog system. *Artificial intelligence* 8, 2 (1977), 155–173.

[9] Dan Bohus and Alexander I. Rudnicky. 2005. Sorry, I didn't catch that!-An investigation of non-understanding errors and recovery strategies. In *6th SIGdial Workshop on Discourse and Dialogue*.

[10] Susan E Brennan. 1998. The grounding problem in conversations with and through computers. *Social and cognitive approaches to interpersonal communication* (1998), 201–225.

[11] Pew Research Center. 2019. Demographics of Mobile Device Ownership and Adoption in the United States. (2019). https://www.pewresearch.org/internet/fact-sheet/mobile/

[12] Fanglin Chen, Kewei Xia, Karan Dhabalia, and Jason I. Hong. 2019. MessageOnTap: A Suggestive Interface to Facilitate Messaging-Related Tasks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article Paper 575, 14 pages. DOI: http://dx.doi.org/10.1145/3290605.3300805

[13] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Unblind Your Apps: Predicting Natural-Language Labels for Mobile GUI Components by Deep Learning. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*.

[14] Janghee Cho and Emilee Rader. 2020. The Role of Conversational Grounding in Supporting Symbiosis Between People and Digital Assistants. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW1 (May 2020).

[15] Herbert H. Clark and Susan E. Brennan. 1991. Grounding in communication. In *Perspectives on socially shared cognition*. APA, Washington, DC, US, 127–149. DOI:http://dx.doi.org/10.1037/10096-006

[16] Benjamin R. Cowan, Nadia Pantidi, David Coyle, Kellie Morrissey, Peter Clarke, Sara Al-Shehri, David Earley, and Natasha Bandeira. 2017. "What Can I Help You with?": Infrequent Users' Experiences of Intelligent Personal Assistants. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '17)*. ACM, New York, NY, USA, Article 43, 12 pages. DOI: http://dx.doi.org/10.1145/3098279.3098539

[17] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 845–854. DOI: http://dx.doi.org/10.1145/3126594.3126651

[18] Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction Mining Mobile Apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 767–776. DOI: http://dx.doi.org/10.1145/2984511.2984581

[19] 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).

[20] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*. 1025–1035.

[21] Thomas RG Green. 1989. Cognitive dimensions of notations. *People and computers V* (1989), 443–460.

[22] 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*. 1–11.

[23] Eric Horvitz. 1999. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 159–166. DOI: http://dx.doi.org/10.1145/302979.303030

[24] Edwin L Hutchins, James D Hollan, and Donald A Norman. 1986. Direct manipulation interfaces. (1986).

[25] Mohit Jain, Pratyush Kumar, Ishita Bhansali, Q Vera Liao, Khai Truong, and Shwetak Patel. 2018a. FarmChat: A Conversational Agent to Answer Farmer Queries. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 170.

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

[27] 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*. ACM, 143–152.

[28] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. 2018. Why Are They Collecting My Data? Inferring the Purposes of Network Traffic in Mobile Apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 1–27.

[29] Daniel Jurafsky and James H Martin. 2019. Dialogue Systems and Chatbots. *Speech and Language Processing* (2019).

[30] James R. Kirk and John E. Laird. 2019. Learning Hierarchical Symbolic Representations to Support Interactive Task Learning and Knowledge Transfer. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. 6095–6102. DOI: http://dx.doi.org/10.24963/ijcai.2019/844

[31] Kazutaka Kurihara, Masataka Goto, Jun Ogata, and Takeo Igarashi. 2006. Speech pen: predictive handwriting based on ambient multimodal recognition. In *Proceedings of the SIGCHI conference on human factors in computing systems (CHI '06)*. ACM, 851–860.

[32] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: Automating & Sharing How-to Knowledge in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1719–1728. DOI: http://dx.doi.org/10.1145/1357054.1357323

[33] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541* (2016).

[34] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. 2017. Adversarial Learning for Neural Dialogue Generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. ACL, Copenhagen, Denmark, 2157–2169. DOI:http://dx.doi.org/10.18653/v1/D17-1230

[35] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6038–6049. DOI: http://dx.doi.org/10.1145/3025453.3025483

[36] Toby Jia-Jun Li, Igor Labutov, Xiaohan Nancy Li, Xiaoyi Zhang, Wenze Shi, Tom M. Mitchell, and Brad A. Myers. 2018a. APPINITE: A Multi-Modal Interface for Specifying Data Descriptions in Programming by Demonstration Using Verbal Instructions. In *Proceedings of the 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2018)*. DOI: http://dx.doi.org/10.1109/VLHCC.2018.8506506

[37] Toby Jia-Jun Li, Igor Labutov, Brad A. Myers, Amos Azaria, Alexander I. Rudnicky, and Tom M. Mitchell. 2018b. Teaching Agents When They Fail: End User Development in Goal-oriented Conversational Agents. In *Studies in Conversational UX Design*. Springer. DOI: http://dx.doi.org/10.1007/978-3-319-95579-7_6

[38] Toby Jia-Jun Li, Tom Mitchell, and Brad Myers. 2020. Interactive Task Learning from GUI-Grounded Natural Language Instructions and Demonstrations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations (ACL 2020)*. ACL, 215–223. https://www.aclweb.org/anthology/2020.acl-demos.25

[39] 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 (UIST 2019)*. ACM. DOI: http://dx.doi.org/10.1145/3332165.3347899

[40] Toby Jia-Jun Li and Oriana Riva. 2018. KITE: Building conversational bots from mobile apps. In *Proceedings of the 16th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2018)*. ACM. DOI:http://dx.doi.org/10.1145/3210240.3210339

[41] Toby Jia-Jun Li, Shilad Sen, and Brent Hecht. 2014. Leveraging Advances in Natural Language Processing to Better Understand Tobler's First Law of Geography. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '14)*. ACM, New York, NY, USA, 513–516. DOI: http://dx.doi.org/10.1145/2666310.2666493

[42] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A. Myers. 2019. Unakite: Scaffolding Developers' Decision-Making Using the Web. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. ACM, New York, NY, USA, 67–80. DOI: http://dx.doi.org/10.1145/3332165.3347908

[43] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning Design Semantics for Mobile Apps. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18)*. ACM, New York, NY, USA, 569–579. DOI: http://dx.doi.org/10.1145/3242587.3242650

[44] 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 (CHI '16)*. ACM, New York, NY, USA, 5286–5297. DOI: http://dx.doi.org/10.1145/2858036.2858288

[45] Jennifer Mankoff, Gregory D Abowd, and Scott E Hudson. 2000. OOPS: a toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces. *Computers & Graphics* 24, 6 (2000), 819–834.

[46] Michael McTear, Ian O'Neill, Philip Hanna, and Xingkun Liu. 2005. Handling errors and determining confirmation strategies—An object-based approach. *Speech Communication* 45, 3 (2005), 249 – 269. DOI: http://dx.doi.org/10.1016/j.specom.2004.11.006 Special Issue on Error Handling in Spoken Dialogue Systems.

[47] Brad A. Myers, Amy J. Ko, Chris Scaffidi, Stephen Oney, YoungSeok Yoon, Kerry Chang, Mary Beth Kery, and Toby Jia-Jun Li. 2017. Making End User Development More Natural. In *New Perspectives in End-User Development*. Springer, Cham, 1–22. DOI: http://dx.doi.org/10.1007/978-3-319-60291-2_1

[48] Chelsea Myers, Anushay Furqan, Jessica Nebolsky, Karina Caro, and Jichen Zhu. 2018. Patterns for how users overcome obstacles in voice user interfaces. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–7.

[49] Don Norman. 2013. *The design of everyday things: Revised and expanded edition*. Basic books.

[50] Sharon Oviatt. 1999a. Mutual disambiguation of recognition errors in a multimodel architecture. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 576–583.

[51] Sharon Oviatt. 1999b. Ten Myths of Multimodal Interaction. *Commun. ACM* 42, 11 (Nov. 1999), 74–81. DOI:http://dx.doi.org/10.1145/319382.319398

[52] Panupong Pasupat, Tian-Shun Jiang, Evan Liu, Kelvin Guu, and Percy Liang. 2018. Mapping natural language commands to web elements. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. ACL, Brussels, Belgium, 4970–4976. DOI: http://dx.doi.org/10.18653/v1/D18-1540

[53] Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (EMNLP '15)*. http://arxiv.org/abs/1508.00305 arXiv: 1508.00305.

[54] Martin Porcheron, Joel E. Fischer, Stuart Reeves, and Sarah Sharples. 2018. Voice Interfaces in Everyday Life. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article Paper 640, 12 pages. DOI: http://dx.doi.org/10.1145/3173574.3174214

[55] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. ACL. http://arxiv.org/abs/1908.10084

[56] Ritam Sarmah, Yunpeng Ding, Di Wang, Cheuk Yin Phipson Lee, Toby Jia-Jun Li, and Xiang 'Anthony' Chen. 2020. Geno: a Developer Tool for Authoring Multimodal Interaction on Existing Web Applications. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST 2020)*.

[57] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *The 30th AAAI Conference on Artificial Intelligence (AAAI '16)*.

[58] 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*. 22–32.

[59] Lanbo She and Joyce Chai. 2017. Interactive Learning of Grounded Verb Semantics towards Human-Robot Communication. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (ACL '17)*. ACL, Vancouver, Canada, 1634–1644. DOI: http://dx.doi.org/10.18653/v1/P17-1150

[60] Ben Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (Aug. 1983), 57–69. DOI: http://dx.doi.org/10.1109/MC.1983.1654471

[61] Shashank Srivastava, Igor Labutov, and Tom Mitchell. 2017. Joint concept learning and semantic parsing from natural language explanations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 1527–1536.

[62] Simone Stumpf, Vidya Rajaram, Lida Li, Margaret Burnett, Thomas Dietterich, Erin Sullivan, Russell Drummond, and Jonathan Herlocker. 2007. Toward Harnessing User Feedback for Machine Learning. In *Proceedings of the 12th International Conference on Intelligent User Interfaces (IUI '07)*. ACM, New York, NY, USA, 82–91. DOI: http://dx.doi.org/10.1145/1216295.1216316

[63] Bernhard Suhm, Brad Myers, and Alex Waibel. 2001. Multimodal Error Correction for Speech User Interfaces. *ACM Trans. Comput.-Hum. Interact.* 8, 1 (March 2001), 60–98. DOI:http://dx.doi.org/10.1145/371127.371166

[64] 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 (EdgeSys'18)*. ACM, New York, NY, USA, 49–54. DOI:http://dx.doi.org/10.1145/3213344.3213353

[65] Geraldine P Wallach and Katharine G Butler. 1994. *Language learning disabilities in school-age children and adolescents: Some principles and applications*. Allyn & Bacon.

[66] Yu Wu, Wei Wu, Chen Xing, Can Xu, Zhoujun Li, and Ming Zhou. 2019. A Sequential Matching Framework for Multi-Turn Response Selection in Retrieval-Based Chatbots. *Computational Linguistics* 45, 1 (March 2019), 163–197. DOI: http://dx.doi.org/10.1162/coli_a_00345

[67] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Michael L Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2017. Toward human parity in conversational speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25, 12 (2017), 2410–2423.

[68] Xiaoyi Zhang, Anne Spencer Ross, Anat Caspi, James Fogarty, and Jacob O. Wobbrock. 2017. Interaction Proxies for Runtime Repair and Enhancement of Mobile Application Accessibility. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6024–6037. DOI: http://dx.doi.org/10.1145/3025453.3025846