DoThisHere: Multimodal Interaction to Improve Cross-Application Tasks on Mobile Devices

Jackie (Junrui) Yang
Stanford University
Stanford, CA, USA
jackiey@cs.stanford.edu

Monica S. Lam
Stanford University
Stanford, CA, USA
lam@cs.stanford.edu

James A. Landay
Institute for Human-Centered AI
Stanford University
Stanford, CA, USA
landay@cs.stanford.edu

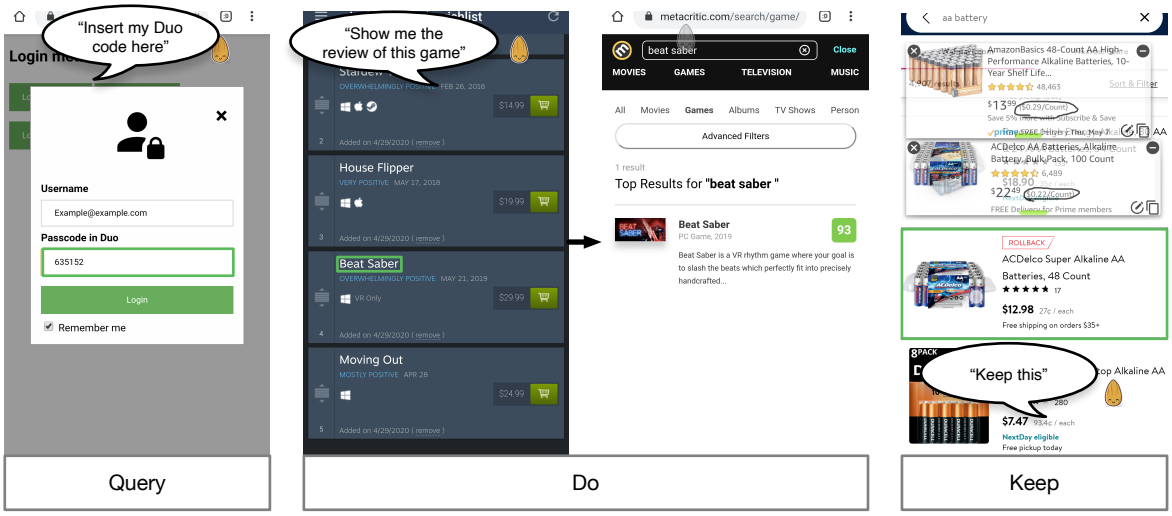

Figure 1. DoThisHere accepts users' multimodal input and helps users complete cross-app tasks. The user can use voice commands (shown in quotes) to specify their intention and touch (shown in green boxes) to specify the relevant parameters. DoThisHere supports three major functions: *Query*, *Do*, and *Keep*, which help the user retrieve information from another app, execute an app function with the current content as a parameter, and remember content on the screen, respectively.

ABSTRACT

Many computing tasks, such as comparison shopping, two-factor authentication, and checking movie reviews, require using multiple apps together. On large screens, "windows, icons, menus, pointer" (WIMP) graphical user interfaces (GUIs) support easy sharing of content and context between multiple apps. So, it is straightforward to see the content from one application and write something relevant in another application, such as looking at the map around a place and typing walking instructions into an email. However, although today's smartphones also use GUIs, they have small screens and limited windowing support, making it hard to switch contexts and exchange data between apps.

We introduce DoThisHere, a multimodal interaction technique that streamlines cross-app tasks and reduces the burden these tasks impose on users. Users can use voice to refer to information or app features that are off-screen and touch to specify where the relevant information should be inserted or is displayed. With DoThisHere, users can access information *from* or carry information *to* other apps with less context switching.

We conducted a survey to find out what cross-app tasks people are currently performing or wish to perform on their smartphones. Among the 125 tasks that we collected from 75 participants, we found that 59 of these tasks are not well supported currently. DoThisHere is helpful in completing 95% of these unsupported tasks. A user study, where users are shown the list of supported voice commands when performing a representative sample of such tasks, suggests that DoThisHere may reduce expert users' cognitive load; the *Query* action, in particular, can help users reduce task completion time.

Author Keywords

Multimodal interaction; voice interfaces; cross-app tasks.

CCS Concepts

•**Human-centered computing** → **Graphical user interfaces; Natural language interfaces; Gestural input;**

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

ACM ISBN 978-1-4503-7514-6/20/10.
http://dx.doi.org/10.1145/3379337.3415841

INTRODUCTION

"There's an app for everything"[1]; however, many tasks require multiple apps to complete them [6]. On desktop or laptop environments, working with multiple apps is relatively easy as the WIMP system allows users to lay out windows side by side and exchange data without disturbing the graphical layout. However, mobile devices have limited screen real estate, and to make it worse, the fat finger problem [4] requires mobile devices to have larger control widgets. Therefore, mobile devices usually only support one app running in the foreground, which makes completing cross-app tasks significantly harder.

Previous work tries to solve this problem by allowing two apps to be laid out side-by-side [1] or improving the task switching interface [8, 13]. However, to make mobile phone apps touch-friendly, they usually use large control widgets, and therefore the information density is more sparse as compared to desktop applications. Combined with the mobile phones' small screen real estate, the amount of information that can be presented is limited. Instead of presenting more information on the screen, DoThisHere allows the user to specify the relevant off-screen information by making verbal references to it and only displays the most relevant information on the screen. Therefore, DoThisHere can potentially reduce both the amount of context switching and the cognitive load on the user.

We identified two types of data exchange that often occur between apps: simple data exchange (copy-and-paste on the desktop) and complex data exchange (looking at and interpreting the information from one app while working on another). For the first type, the user may either refer to the relevant information verbally and specify the destination with touch (*Query* action in Figure 1), or specify the relevant information with touch and refer to the target action in a verbal command (*Do* action in Figure 1). For the second type, we allow the system to capture part of the UI of an app and the user can put it on top of another app like a "post-it note" (*Keep* action in Figure 1).

It is important that we make DoThisHere applicable to apps without requiring any modification. We do so by leveraging the APIs provided by operating systems (OSs) and an open-source virtual assistant framework. We use mobile OS APIs for recognizing information and input control on-screen, intercepting the user's touch points for selection, and displaying relevant results. For understanding the user's voice commands, retrieving off-screen information, and executing the user-specified command, we leverage the speech understanding and app integration functionality of a virtual assistant framework. In this way, DoThisHere can take advantage of the large number of apps that have already been integrated into the virtual assistant framework; DoThisHere can continue to expand its functionality as more apps are integrated into the assistant.

The contributions of DoThisHere include:

1. A multimodal interaction paradigm that facilitates completing cross-app tasks on mobile devices by letting users make verbal references to off-screen content and app functions.
2. A system design and implementation that is portable and expandable as the virtual assistant framework grows.
3. An evaluation showing that the DoThisHere system can apply to the majority of cross-app tasks that are currently not well-supported by mobile OSs. We compared the maximum performance of our system with Android navigation in actual cross-app tasks with a user study. When users are provided with the list of supported voice commands, our system induces less task load (based on NASA-TLX) on the user, the majority of users prefer DoThisHere over the baseline Android navigation, and the *Query* action in DoThisHere can significantly improve the user's performance time in completing tasks.

[1] https://www.wired.com/2010/10/app-for-that/

RELATED WORK

There are three categories of related work for DoThisHere: multimodal virtual assistant systems, window managers that support voice, and research on small screen multi-tasking.

Multimodal virtual assistant

DoThisHere is built upon voice interactions provided by the Almond virtual assistant framework [7]. We added touch interactions to specify which GUI elements to evoke verbally specified actions upon. There is other research in this area of building a virtual assistant that can receive multimodal input. MVA [15] is one of the earliest projects in this area. It proposed a system similar to existing smartphone virtual assistants but can also receive input from touch and voice to better define information queries. However, MVA requires a complete rebuild of the original virtual assistant infrastructure to make commands that can accept input from both channels. Brassau [12] makes the process of adopting multiple modalities easier by automatically generating a GUI to accompany any existing voice command. Similar to DoThisHere, these systems use virtual assistant frameworks and multimodal interaction, but their goal is to enhance the experience for single-app tasks that are supported by voice commands, while DoThisHere is aimed to reduce a user's task load while performing cross-app tasks that involve multiple GUI applications.

Other researchers have also built assistive systems for mobile phones that leverage multimodal interaction to facilitate UI automation. Li et al. have proposed two systems, SUGILITE [18] and APPINITE [19], that can help virtual assistants learn commands and concepts through multimodal input from users. Similarly, Gesto [21] allows users to record UI commands and evoke them by voice or gesture. These three systems work similarly to DoThisHere in that they use voice to understand the user's intention and touch to understand the UI element that the user wants to manipulate, but for the different purpose of allowing end-users to expand virtual assistant skills.

Window managers with multimodal input

Another category of related work focuses on building screen managers with different modalities including voice. Using voice to control computers is not a new idea. Richard Bolt [5] demonstrated an interface that can accept voice and gesture as input as early as 1980. In 1993, Bellik et al. [3] brought this

idea to a window manager that allows apps built within their framework to accept voice, touch, and traditional mouse clicks as input. Cohen et al. [10] presented the QuickSet system that uses multimodal interaction for distributed applications, but the interaction paradigm was not yet integrated at an OS-level. More recently, on smartphones, Cutugno et al. [11] have proposed a framework for developers to build stand-alone apps that can accept multimodal input, but their framework does not support interaction across different applications. These systems primarily focused on allowing users to execute a single action via different modalities. To support multimodal input, they require apps to be rebuilt with their framework and they have not demonstrated benefits for completing cross-app tasks.

Other publications on window managers are mainly focused on using voice as the single channel of input. For example, Odell et al. [20] described the architecture of the built-in voice control system for Windows Vista. People have also implemented similar systems for mobile and web platforms. JustSpeak [22] supports using voice commands as a replacement for touch to interact with Android apps through system-supported accessibility APIs. Capti-speak [2] is a similar voice control system, but for web interfaces. Although these systems allow users to use voice to manipulate GUI interfaces, they are primarily built for accessibility purposes. As every voice command is almost always mapped to a single GUI action, the amount of context switching and information to remember remains the same as compared to pure GUI interactions. Therefore, these systems are unlikely to reduce users' mental and physical load on cross-app tasks.

Small-screen multi-app usage

There is a range of prior work that aims to support multi-app usage on small screens. Some collected data from users' smartphone usage traces to better describe people's existing behavior regarding multi-app usage [6, 9, 16, 17]. Others built techniques to better facilitate usage across different apps. Almost all smartphones these days have some kind of app switcher to allow users to switch back-and-forth between apps in the foreground [2]. Some Android devices also have a split-screen mode [1] that lets users see two applications at the same time. As a research project, Peek-a-View [8] helps the user switch between a messaging app and another app by using interactions with the screen cover. Gupta et al. [13] built a system that recognizes different fingers of the user's hand and lets the user interact with two applications that are overlaid on top of each other at the same time. The former is focused on task switching between two unrelated apps (messaging app and other apps) while the latter requires apps to be specially built for this interaction. In comparison, DoThisHere focuses on making apps work together to support the same task and does not require that the apps be specially built.

DOTHISHERE CONCEPTS

We identified data exchange as one of the important challenges for completing tasks on mobile devices that involve multiple apps. The data exchanged can be a complex piece of information (e.g., write an email about walking instructions between two locations), or a simple piece of text that needs to be shared (e.g., insert my remaining travel budget from Mint into Google Flights). This data exchange can happen within the same app (e.g., compare two restaurants in yelp) or across different apps (e.g., find the IMDB review of a movie on fandango). Also, the data exchange can happen now (e.g., write an email about the news that I just read) or can span a period of time (e.g., write an email about an item that I saw last week on Amazon). We wanted DoThisHere to be able to handle all of these different types of data exchange consistently.

To allow users to access this new functionality, we float the DoThisHere icon on top of apps; users can tap on it to start DoThisHere listening for voice commands and monitoring for screen selections.

Query

On the desktop, simple data exchange between apps is usually handled by "copy" and "paste." In copy and paste actions, one app is functioning as a content provider and the other app is acting as a content receiver. On mobile devices, the screen size is limited and the application navigation hierarchy is likely to be deeper. Finding the desired information or going to the desired feature is a much harder problem. To solve this, we introduce a "*Query*" feature (shown in Figure 1a), so that the user can get a piece of (usually off-screen) information from another app specified by voice and insert it back into a textbox specified by touch in the current app without needing to switch apps.

Do

We also support a "*Do*" feature, so that the user can send a piece of information in the current app to a feature off-screen in another app specified by voice without needing to copy the information and manually find that feature. In other words, *Query* can help users in simple information retrieval when they are in the content *receiver* app, whereas *Do* can help users who are already in the content *provider* app.

Keep

For complex data, it is useful to see the content from one app while working in another. On desktop devices, this can be achieved by laying out multiple windows side-by-side. However, on mobile devices, the small screen can only fit a limited amount of content, and apps usually have large control widgets for finger touch, which further reduces the possibility of presenting information efficiently.

As the user usually only needs to see the relatively static information displayed in the source app while interactively working in another app, it is possible to improve complex data exchange by only showing the important piece of information in the source app while allowing the user to interact with another app using almost the entire screen. Thus, we introduce the "*Keep*" concept. *Keep* allows users to snip a piece of content on the screen and keep it visible while using another app. The user indicates the area of interest by selecting the content and saying "keep this" to DoThisHere. That area

[2] *Recents screen* introduced in Android 5.0: https://developer.android.com/guide/components/activities/recents

is shown on the screen like a "post-it note." The user can drag it around, make it smaller or larger by using two-finger pinching, minimize it to an icon, add annotations to it by tapping on the pencil icon, and close it when needed. If the user wishes to refer to a piece of content at a later time, they can ask DoThisHere to remember it, without keeping it on the screen, by saying "remember this as [tag]." When they need the information again, they can retrieve the off-screen content by saying "recall the [tag]."

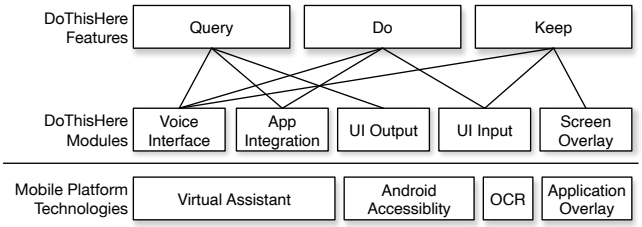

Figure 2. DoThisHere uses existing technologies provided by mobile platforms and apps, so that its features can be easily expanded to new platforms and apps.

SYSTEM ARCHITECTURE

As the DoThisHere concepts deal with interactions between apps, they need to be incorporated at the operating system level. We leverage existing technologies on mobile platforms so DoThisHere can work with existing apps without modification. As shown in our system architecture (see Figure 2), the *Query*, *Do*, and *Keep* features are built on top of five functional modules, which in turn are based on four mobile platform technologies.

Voice interface and app integration

DoThisHere allows users to use voice to specify which off-screen information they want to use, and what app functionality they want to trigger in the *Query* and *Do* commands, respectively. To achieve this, we leverage an existing virtual assistant that has been designed to perform a large number of skills. Our system uses Almond, an open-source virtual assistant [7]. Almond's speech understanding translates natural language into a domain-specific language (DSL) called *ThingTalk*.

We expanded *ThingTalk* to accept multimodal input by adding a primitive for getting the content on the screen (reading text), and a function for outputting results on the screen (writing to textboxes). We also added new skills for the *Keep* feature: keep, remember, and recall.

The *Query* and *Do* commands depend on third-party apps actively exposing their in-app information and APIs to DoThisHere (app integration). Getting app integration could be hard if we have to start from scratch. Luckily, many apps have already exposed their functions to virtual assistants and those functions are exactly what DoThisHere needs. For our implementation of DoThisHere we leveraged app integrations provided by Almond. As described above, by extending the DSL, the user can use screen content instead of voice as input parameters, and textboxes on-screen as output instead of voice feedback when using the app-provided features. All of this can be done without any additional app modification.

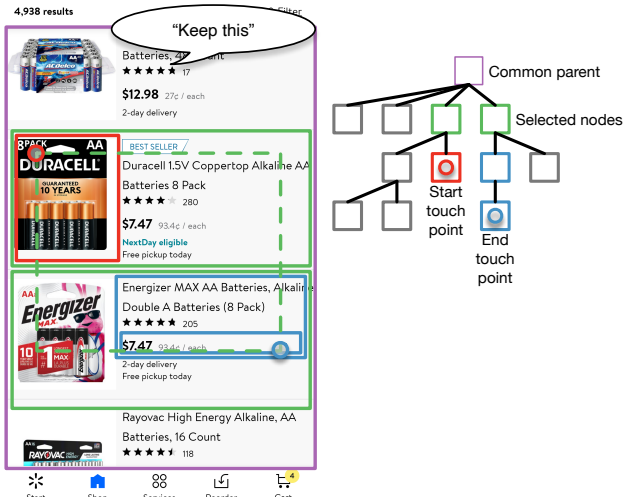

Figure 3. DoThisHere supports flexible and robust UI node selection. The red circle is the user's start touchpoint and the blue circle is the user's end touchpoint. Each solid-line box represents a UI node, and the tree represents the UI's hierarchical structure.

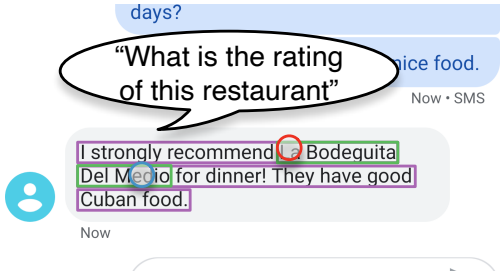

Figure 4. DoThisHere supports partial text selection on screen by OCR. The red circle is the user's start touchpoint and the blue circle is the user's end touchpoint. The purple boxes are recognized text and the green boxes are the user-selected text.

UI selection

To automatically get and set content on the screen for app integrations and remembering things, we built the DoThisHere system on top of the Android accessibility APIs. From the OS, we can get the list of UI nodes that are currently on display, their hierarchical structure, and their content. Using the UI nodes, we created an interface so that users can easily select parts of the UI for the *Query*, *Do*, and *Keep* actions.

We want our selection UI to be flexible and resistant to error. It should accommodate different kinds of content that the user wishes to select; it should be easy to perform on an error-prone touch screen. However, if we ask the user to directly select the desired area, it can be challenging for them to precisely specify the bounding box of the area on a mobile device. For example, as shown in Figure 3, the user wants to select both battery listings in the search result, but they have not dragged the bounding box all the way through. The green dotted-line box represents the bounding box that the user has dragged across, while the two green solid-line boxes are the two battery listings that the user intends to select.

Therefore, DoThisHere leverages the hierarchical structure of UI nodes (as shown in Figure 3). For example, the start touch point is within the red node, the green node, and the purple node, and the end touch point is within the blue nodes, the green node, and the purple node. One way to implement this selection algorithm is to select every bottom-level node that has an intersection with the user's selection box (shown by the green dotted line). This method is flexible but would have missed the delivery information in the second battery listing. Another more error-resistant approach is to find the common ancestor that contains both the start touchpoint and the end touchpoint, but this method is not flexible enough to select only the two battery listings. Instead, it would select the entire area of the purple node.

DoThisHere solves the problems of both solutions by first finding the common ancestor (purple box), and then selects all the nodes that are direct children of that ancestor who have intersected with the selection box. In this way, the system can precisely find the two green nodes even if the user's touchpoints are not entirely accurate.

For *Query* commands, we noticed that sometimes, UI node-level selection is not enough. For example, in Figure 4, the user wants to know the rating of a restaurant, and wants to select only a part of the text in that node. However, the entire text message is shown as one node in the accessibility API. To solve this, we use an OCR library to recognize every line of words on the screen. If the user's start and end touchpoints are both on the same group of recognized text (shown in purple boxes), we treat the start and the end touchpoints as the beginning and the end of the text selection (shown in green boxes) and extract that text for the executed command. Note that this also has the added benefit of allowing the user to select text in images.

Screen overlay

DoThisHere needs to provide a special UI that can be displayed on top of other apps. We implemented DoThisHere's display component as an application overlay in Android. The overlay API can be used to draw boxes of selected UI content to provide real-time feedback for the user during the selection process. We also rely upon the overlay API to draw the user's kept UI content as a half-transparent layer so that it is less disturbing for the user to use the current foreground app. We also use this same API to display the Almond icon for triggering commands as mentioned above.

Implementation

In our current version of DoThisHere, we use speech recognition from Microsoft Azure, and natural language parsing from Almond with the addition of some regular expression templates. In this way, DoThisHere can parse common commands accurately, while leveraging the full vocabulary of the Almond virtual assistant. DoThisHere relies on Almond to integrate with other apps. During the development of DoThisHere, we also built a few demo apps (a budget app, a two-factor authentication app, and a password manager) and integrated them into Almond for our user study.

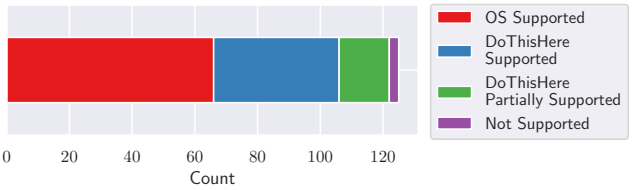

Figure 5. We found that DoThisHere can support almost all of the tasks that are not well-supported by current mobile operating systems.

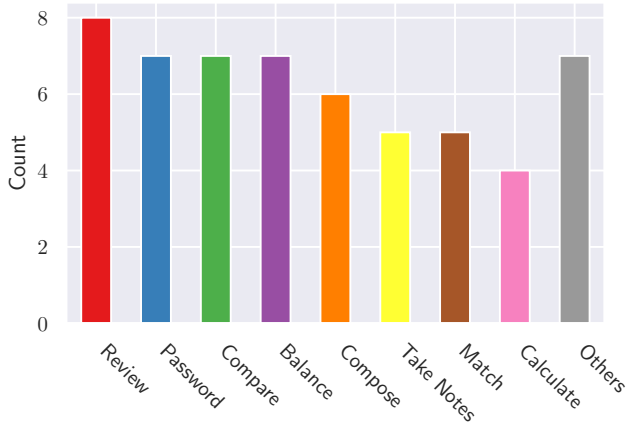

Figure 6. The most common categories of tasks found in our survey that are supported or partially supported by DoThisHere.

All features of DoThisHere can be packaged as a regular Android application; there is no need to root the device. Due to the modular design of DoThisHere, we expect DoThisHere to be portable to other operating systems if similar APIs are available. As the virtual assistant framework gains more apps, DoThisHere can also get these additional apps for free.

EVALUATION

We conducted two studies to design and evaluate DoThisHere. In the first study, we gathered common cross-app tasks performed on smartphones and evaluated whether DoThisHere could be helpful for those tasks. In the second study, we asked participants to perform a representative sample of those tasks and compared their performance and stress levels when performing those tasks with and without DoThisHere.

Study 1: How people use cross-app tasks

To design DoThisHere, we first investigated how people utilize multiple apps to accomplish a single task on their smartphones. Prior work by Böhmer et al. [6] showed that even back in 2011, 31.8% of smartphone usage sessions involved two or more apps. However, they did not investigate whether those apps were used to accomplish the same or different tasks. To know more about users' cross-app usage, we conducted a survey asking about tasks people are performing or wish to perform that involve multiple apps working together, and the apps involved.

We conducted the survey on Amazon Mechanical Turk and collected 125 valid[3] responses from 75 unique participants. We

[3]We removed invalid tasks in the following categories: (1) tasks with unclear descriptions (e.g., "To do daily mobile work"), (2) tasks that

Category	Example
Review	Find a show on Hulu with a good IMDB rating.
Password	Login to the corporate Outlook server with Microsoft Authenticator.
Compare	Find prices of groceries on Fred Meyer and compare them to those at Safeway.
Balance	Buy crypto currencies on Coinbase with a portion of the bank balance from a Wells Fargo account.
Compose	Compose directions to a bar in Messages while looking at the map in Google Maps.
Take Notes	Create a to-do item in Reminders from content in Messages.
Match	Find the best time for hiking by matching the traffic data from Google Maps and the weather forecast from the Weather app.
Calculate	Calculate tips with Calculator for a DoorDash delivery.

Table 1. Examples of the most common categories of tasks identified in our survey that are supported or partially supported by DoThisHere.

paid participants $1 for a 5-minute survey, which corresponds to a wage of $12 per hour. This study was approved by our university's IRB. One author labeled each task by how well the task is supported by common mobile operating systems and by our proposed DoThisHere architecture. Each task is classified into these four categories:

1. OS supported (66 tasks): The amount of task switching and the load on a user's working memory on a current-generation mobile OS is comparable with that on a desktop OS (e.g., post a photo in the camera roll to a social network; this feature is well supported by the built-in share sheet.)

2. DoThisHere supported (40 tasks): With DoThisHere, the amount of task switching and the load on a user's working memory can be reduced to the same level as on a desktop OS (e.g., set a timer according to a recipe app; the user can select the relevant time and use a *Do* action with a timer app.)

3. DoThisHere partially supported (16 tasks): With DoThisHere, the amount of task switching and the load on a user's working memory is reduced, but not yet to the same level as on a desktop OS (e.g., compare the price of the same item on eBay and Amazon; the user can use a *Keep* action to keep the price of the item and reduce the load on working memory, but cannot browse the two websites side-by-side as on a desktop.)

4. Not supported (3 tasks): DoThisHere cannot be used in this task (e.g., browsing relevant information on a website while on a phone call; DoThisHere cannot extract information from a phone call.)

involve apps of identical, rather than complementary, functionalities (e.g., Temple Run and Candy Crush for "Playing games"), and (3) irrelevant responses (spam).

As shown in Figure 5, a significant portion (59 out of 125) of cross-app tasks are not yet well-supported by mobile OSs. Among these tasks, DoThisHere can potentially help users reduce the amount of task switching and mental load on 56 tasks (95%).

We then used open coding to generate a list of categories for tasks that DoThisHere supports (including partially supported tasks). We conducted open coding on the filtered survey data to develop codes that describe the tasks involved (e.g., choose an item based on reviews). Then we conducted axial coding to group these codes into high-level categories (e.g., review). During the analysis, 28 codes emerged initially from the open coding stage and the final results contain 12 high-level categories. We list the common categories (categories with more than 3 tasks) in Figure 6, and group other categories into *Others* in the figure. We list examples for these common categories in Table 1.

Study 2: Task performance and cognitive load with DoThisHere

To evaluate how well DoThisHere can help users in a real-world task, we conducted a study comparing users' time performance and cognitive load with and without DoThisHere. As a baseline, we compared DoThisHere to the native Android navigation system.

Tasks

To construct a list of user study tasks that are representative of real world experience, we used the top four categories of tasks we identified from Study 1. Those categories covered more than 50% of the collected DoThisHere supported tasks. We designed three sets of tasks shown in Table 2, each containing one task in each of the four selected categories. In the study, we used one task set for training, one task set for the baseline condition, and one task set for the DoThisHere condition in a counter-balanced order.

Procedure

Our study was conducted during the COVID-19 pandemic, and we took measures to follow social-distancing guidelines while evaluating DoThisHere. Although DoThisHere is packaged into a standalone application that can be installed on most Android phones, we also need to set up an environment that includes demo apps and accounts for the user study tasks. Therefore, participants had to have access to our physical device for the evaluation. So we physically delivered the demo device to the participants in a package and conducted the rest of the study remotely. We used ADB over TCP/IP [4] and the Zoom teleconferencing software to achieve a similar format to a normal user study.

We first gave the participant a brief introduction to the interaction for both DoThisHere and the baseline.

We then trained the users for both conditions so that users could get close to their maximum performance for both systems, *even if they are not regular Android users*. We first asked the participants to use the baseline condition to complete the

[4]Connect to ADB over Wi-Fi: https://developer.android.com/studio/command-line/adb#wireless

Task	Category	Feature	Taskset 1	Taskset 2	Taskset 3
1	Compare	*Keep*	Find the cheapest battery on Amazon and Walmart.	Find the most reviewed repair service from Google Maps and Yelp.	Find the cheapest hotel on Hotel.com and Priceline.
2	Password	*Query*	Login to a website with Duo.	Login with SMS.	Login with a password stored in a password manager.
3	Balance	*Query*	Filter Ebay results using a shopping budget.	Filter hotel result by a travel budget.	Buy bitcoin with half of a bank balance.
4	Review	*Do*	Find the Netflix show with the best review in the IMDB.	Find the Amazon book with the best goodreads review.	Find the Steam game with the best review on Metacritic.

Table 2. We derived tasks used in Study 2 from the most common categories of cross-app tasks that are not well-supported by current-gen mobile OSs.

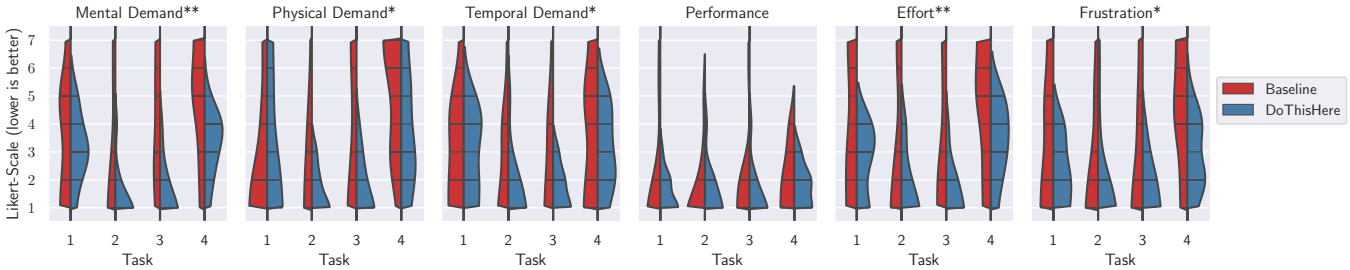

Figure 7. NASA-TLX results for Study 2 showed that DoThisHere can reduce participants' cognitive load while working on cross-app tasks.
This plot compares distributions of the user's Likert-scale response between conditions (left: Baseline, right: DoThisHere) in the NASA-TLX questions for all four tasks. Thicker bands indicates there are more responses of that option.
***: statistically significant (p<0.05), **: (p<0.01)**
Note that the task-wise comparison is done by comparing all NASA-TLX question-task pairs between two conditions for each task.

training task set. During the process, we gave them tips about the baseline condition such as double-tapping the square button on the navigation bar to quickly switch between recent apps. After that, we asked the participants to use DoThisHere to complete the same training task set. During the process, we suggested which features in DoThisHere to use for each task, as shown in Table 2.

After the training session, we asked the participants to complete a unique task set with each condition. So both conditions were less constrained by learning time, in the baseline condition we showed hints about the Android navigation system, and in the DoThisHere condition we showed the list of supported voice commands in the tasks. We timed each task and asked participants to fill out the NASA-TLX [14] form after each task. The order of the task sets and the order of the two conditions were both counter-balanced between participants. After the two conditions, we collected the participants' subjective feedback on both systems.

Participants

We recruited 12 participants (5 female, 7 male), aged 22 to 38 (median 24). Two of them were regular Android users. Each participant was compensated $15 for their participation. The total session time was around 45 minutes. This study was approved by our university's IRB.

Results

The participants' reported task loads are shown in Figure 7. We computed two-sided Wilcoxon tests for the NASA-TLX responses combined for all tasks and corrected the p-value by

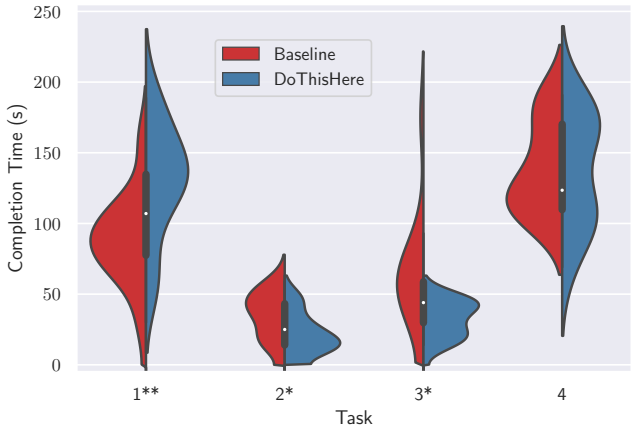

Figure 8. Study 2 showed that DoThisHere reduced task completion time for Task 2 and 3 (*Query*), but increased it for Task 1 (*Keep*).
This plot compares distributions of the user's task completion time between conditions (left: Baseline, right: DoThisHere) for all four tasks. Thicker bands indicates more users finished their tasks at that duration (lower is better).
***: statistically significant (p<0.05), **: (p<0.01)**

Holm's sequential Bonferroni procedure. We found that DoThisHere had a statistically significant positive effect on participant's mental load ($p < 0.001$[5]), physical load ($p = 0.017$),

[5]When p is smaller than 0.001, we report $p < 0.001$, otherwise, we report the approximate value of p up to three significant digits.

temporal demand ($p = 0.017$), effort ($p = 0.004$), and frustration ($p = 0.016$). This shows that, in general, DoThisHere can help participants reduce their task load while doing cross-app tasks.

The task completion times are shown in Figure 8. We computed two-sided paired t-tests for the completion time and corrected the p-value by Holm's sequential Bonferroni procedure. For tasks 2 and 3, we observed a statistically significant reduction ($p = 0.032$ and $p = 0.022$ respectively). This shows that *Query* can help participants reduce their task completion time significantly. For task 1, we observed an *increase* in completion time ($p = 0.002$). We think the reason why participants spent more time in Task 1 with DoThisHere is because remembering the best battery in their mind is faster than interacting with DoThisHere. For task 4, we didn't observe any statistically significant difference in task completion time.

Eight out of 12 participants reported that they preferred the DoThisHere condition to the baseline, three other participants reported that they would prefer DoThisHere in some of the tasks, and one participant said that he preferred not to use DoThisHere. Six participants mentioned that DoThisHere can help them remember things so they don't have to. Five participants mentioned that DoThisHere can reduce or ease the switching between apps. P6 and P11 mentioned that the delay in triggering DoThisHere affected their performance. Participants had mixed feelings about the selection system: P7 and P10 were happy to see that DoThisHere can select text in pictures, while P5, P8, and P11 complained that the selection was not always accurate. We also noticed that even though Task 4 was designed for *Do*, three participants used *Keep* along with *Do* in that task to help them remember things.

DISCUSSION

In this section, we discuss the limitation of our current evaluation and implementation and how future research may solve these issues.

We noticed that *Keep* actions increased the task completion time in Study 2, although many participants explicitly said that they liked this feature and some even used it in tasks that were designed for other DoThisHere features. Due to the ease of remembering something for a short period, we think *Keep* may not be the fastest way to do it in this artificial situation. Triggering the action and performing the selection takes around 10 seconds, which is more than enough time for participants to remember the information manually in a timed user study session. However, in real life, *Keep* should still be helpful when the user needs to remember complex information or remember information for a long time. Delay in the DoThisHere system is another factor that contributes to the longer task completion time. Improving the implementation of DoThisHere can reduce the delay between different functions, especially the speech recognition function, and can potentially improve task performance.

Also, although we implemented the "remember" and "recall" commands for the *Keep* action, we were not able to test them in the user study. One possible use case is for users to remember an interesting piece of news with DoThisHere, and recall it in future conversations with friends. However, such use cases span a longer period of time, and we could not easily test these in the user study.

As stated above, the current implementation of DoThisHere has some delay (approximately 2 seconds) after the user taps on the icon before the system can start listening to voice commands and UI selections. There are a few sources of this delay: The Android screen capture API takes time to get a screenshot, the OCR library takes time to recognize text, and the audio recording function takes time to start. Future systems can try to parallelize these action or start some of those activities before the user triggers a DoThisHere action[6].

DoThisHere uses voice interaction extensively to allow the user to quickly express their intention to complement precise UI selection for the content. Our paper did not focus on the challenges of the speech interaction system, such as handling ambiguity and providing feedback, as these are handled by the virtual assistant framework in our system design. Future work may be able to further optimize the voice assistant framework specifically for DoThisHere interactions.

We noticed that there is still room for improvement in the UI element selection algorithm. Although our algorithm is robust to input error, having to drag to select elements on screen is still slow, and because of the fat-finger problem the user's movement is also at a reduced speed. Future systems may improve on this by inferring a user's intention from the user's voice command and suggesting what the user may want to select so that the user can select the corresponding element in a single touch. For example, the selection has to be a textbox if the user says "Insert my bank balance here" while the selection is likely to be one or more rows in a list if the user says "Keep these items."

Another limitation of our current system is that the kept content from the *Keep* action is completely static. Although users can move it around and annotate it, they cannot interact with the underlying app. We made this design decision because dynamic, interactive content either requires deep OS integration or custom app integration, which limits our portability. Also, DoThisHere leverages the fact that the kept content is not interactive, which allows the text to be scaled down as a way of efficiently using the screen space. However, supporting dynamic kept content can help some tasks to be fully supported by DoThisHere (instead of partly supported). For example, currently, if the user wants to read a book to another person during a video chat, they would have to ask DoThisHere to remember each page of the book separately and ask DoThisHere to recall it later. By supporting dynamic content, the user only needs to ask DoThisHere to remember the (changing) page once. We leave the exploration of supporting dynamic kept content for future work.

CONCLUSION

Our current-generation of mobile OSs offer limited support for cross-app tasks due to small screen sizes. In this paper, we

[6]Shazam keeps the microphone open for a while after the user exits the app so that if it's opened again, it can immediately start recording: https://www.theregister.co.uk/2016/11/15/shazam_listening/

present DoThisHere, a system that uses multimodal interaction to help users access information *from* or carry information *to* other apps with less context switching. We designed the system in a way that is portable to other operating systems and can be expanded to more apps when they are integrated with an existing virtual assistant framework. We designed and evaluated the system through two user studies and discovered that DoThisHere has the potential to help users on almost all of the cross-app tasks that are not well-supported by current mobile operating systems. Our study shows an early indication that DoThisHere may help reduce a user's task load when doing actual cross-app tasks. We believe DoThisHere is a concrete step towards a more capable and natural way of interacting on future smartphones and we strongly encourage mobile OS developers to adopt our interaction techniques.

ACKNOWLEDGEMENT

The authors would like to acknowledge Tianshi Li for her help with the early prototypes and writing. Also, the authors would like to thank Giovanni Campagna and Silei Xu for their help with Almond integration. The authors would also like to thank Gaurab Banerjee for giving early feedback. Finally, the authors would also like to thank the reviewers for their constructive feedback.

This work is supported in part by the National Science Foundation under Grant No. 1900638.

REFERENCES

[1] 2016. Multi-Window Support | Android Developers. https://developer.android.com/guide/topics/ui/multi-window. (2016). (Accessed on 05/05/2020).

[2] Vikas Ashok, Yevgen Borodin, Yury Puzis, and I. V. Ramakrishnan. 2015. Capti-speak: a speech-enabled web screen reader. In *Proceedings of the 12th Web for All Conference on - W4A '15*. ACM Press. DOI: http://dx.doi.org/10.1145/2745555.2746660

[3] Yacine Bellik and Daniel Teil. 1993. A multimodal dialogue controller for multimodal user interface management system application: a multimodal window manager. In *INTERACT '93 and CHI '93 conference companion on Human factors in computing systems - CHI '93*. ACM Press. DOI: http://dx.doi.org/10.1145/259964.260124

[4] Xiaojun Bi, Yang Li, and Shumin Zhai. 2013. FFitts law: modeling finger touch with fitts' law. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*. ACM Press. DOI: http://dx.doi.org/10.1145/2470654.2466180

[5] Richard A. Bolt. 1980. "Put-that-there". *ACM SIGGRAPH Computer Graphics* 14, 3 (Jul 1980), 262–270. DOI: http://dx.doi.org/10.1145/965105.807503

[6] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. 2011. Falling asleep with Angry Birds, Facebook and Kindle: a large scale study on mobile application usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services - MobileHCI '11*. ACM Press. DOI: http://dx.doi.org/10.1145/2037373.2037383

[7] Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S. Lam. 2017. Almond: The Architecture of an Open, Crowdsourced, Privacy-Preserving, Programmable Virtual Assistant. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. DOI: http://dx.doi.org/10.1145/3038912.3052562

[8] Koeun Choi, Hyunjoo Song, Kyle Koh, Jinwook Bok, and Jinwook Seo. 2016. Peek-a-View: Smartphone Cover Interaction for Multi-Tasking. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM. DOI: http://dx.doi.org/10.1145/2858036.2858426

[9] Leor Cohen. 2015. World attending in interaction: Multitasking, spatializing, narrativizing with mobile devices and Tinder. *Discourse, Context & Media* 9 (sep 2015), 46–54. DOI: http://dx.doi.org/10.1016/j.dcm.2015.08.001

[10] Philip R. Cohen, Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen, and Josh Clow. 1997. QuickSet. *Proceedings of the fifth ACM international conference on Multimedia - MULTIMEDIA '97* (1997). DOI: http://dx.doi.org/10.1145/266180.266328

[11] Francesco Cutugno, Vincenza Anna Leano, Roberto Rinaldi, and Gianluca Mignini. 2012. Multimodal framework for mobile interaction. *Proceedings of the International Working Conference on Advanced Visual Interfaces - AVI '12* (2012). DOI: http://dx.doi.org/10.1145/2254556.2254592

[12] Michael Fischer, Giovanni Campagna, Silei Xu, and Monica S. Lam. 2018. Brassau: automatic generation of graphical user interfaces for virtual assistants. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM. DOI: http://dx.doi.org/10.1145/3229434.3229481

[13] Aakar Gupta, Muhammed Anwar, and Ravin Balakrishnan. 2016. Porous Interfaces for Small Screen Multitasking using Finger Identification. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM. DOI: http://dx.doi.org/10.1145/2984511.2984557

[14] Sandra G. Hart. 2006. Nasa-Task Load Index (NASA-TLX): 20 Years Later. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 50, 9 (oct 2006), 904–908. DOI: http://dx.doi.org/10.1177/154193120605000909

[15] Michael Johnston, John Chen, Patrick Ehlen, Hyuckchul Jung, Jay Lieske, Aarthi Reddy, Ethan Selfridge, Svetlana Stoyanchev, Brant Vasilieff, and Jay Wilpon. 2014. MVA: The Multimodal Virtual Assistant. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. Association for Computational Linguistics. DOI: http://dx.doi.org/10.3115/v1/w14-4335

[16] Simon L. Jones, Denzil Ferreira, Simo Hosio, Jorge Goncalves, and Vassilis Kostakos. 2015. Revisitation analysis of smartphone app use. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '15*. ACM Press. DOI:http://dx.doi.org/10.1145/2750858.2807542

[17] Luis Leiva, Matthias Böhmer, Sven Gehring, and Antonio Krüger. 2012. Back to the app: the costs of mobile application interruptions. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services - MobileHCI '12*. ACM Press. DOI: http://dx.doi.org/10.1145/2371574.2371617

[18] 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*. ACM. DOI: http://dx.doi.org/10.1145/3025453.3025483

[19] 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. DOI: http://dx.doi.org/10.1109/vlhcc.2018.8506506

[20] Julian Odell and Kunal Mukerjee. 2007. Architecture, User Interface, and Enabling Technology in Windows Vista's Speech Systems. *IEEE Trans. Comput.* 56, 9 (sep 2007), 1156–1168. DOI: http://dx.doi.org/10.1109/tc.2007.1065

[21] Chang Min Park, Taeyeon Ki, Ali J. Ben Ali, Nikhil Sunil Pawar, Karthik Dantu, Steven Y. Ko, and Lukasz Ziarek. 2019. Gesto: Mapping UI Events to Gestures and Voice Commands. *Proceedings of the ACM on Human-Computer Interaction* 3, EICS (jun 2019), 1–22. DOI:http://dx.doi.org/10.1145/3300964

[22] 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 on - W4A '14*. ACM Press. DOI:http://dx.doi.org/10.1145/2596695.2596720