App-Based Task Shortcuts for Virtual Assistants

Deniz Arsan
University of Illinois at Urbana-Champaign
Urbana, IL, USA
darsan2@illinois.edu

Ali Zaidi
University of Illinois at Urbana-Champaign
Urbana, IL, USA
aliz2@illinois.edu

Aravind Sagar
UserTesting Inc.
San Francisco, CA, USA
asagar@usertesting.com

Ranjitha Kumar
University of Illinois at Urbana-Champaign
Urbana, IL, USA
ranjitha@illinois.edu

ABSTRACT

Virtual assistants like Google Assistant and Siri often interface with external apps when they cannot directly perform a task. Currently, developers must manually expose the capabilities of their apps to virtual assistants, using *App Actions* on Android or *Shortcuts* on iOS. This paper presents SAVANT, a system that automatically generates task shortcuts for virtual assistants by mapping user tasks to relevant UI screens in apps. For a given natural language task (e.g., "send money to Joe"), SAVANT leverages text and semantic information contained within UIs to identify relevant screens, and intent modeling to parse and map entities (e.g., "Joe") to required UI inputs. Therefore, SAVANT allows virtual assistants to interface with apps and handle new tasks without requiring any developer effort. To evaluate SAVANT, we performed a user study to identify common tasks users perform with virtual assistants. We then demonstrate that SAVANT can find relevant app screens for those tasks and autocomplete the UI inputs.

CCS CONCEPTS

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

KEYWORDS

Virtual assistants, mobile apps, interaction mining, task shortcuts

ACM Reference Format:
Deniz Arsan, Ali Zaidi, Aravind Sagar, and Ranjitha Kumar. 2021. App-Based Task Shortcuts for Virtual Assistants. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21), October 10–14, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3472749.3474808

UIST '21, October 10–14, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8635-7/21/10…$15.00
https://doi.org/10.1145/3472749.3474808

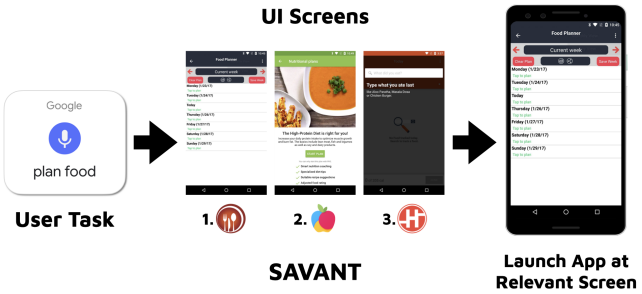


Figure 1: SAVANT automatically generates task shortcuts for virtual assistants by mapping user tasks to relevant UI screens in apps.

1 INTRODUCTION

Virtual assistants like Google Assistant, Siri, and Alexa automate a variety of everyday tasks. While the set of tasks that virtual assistants can perform is growing every year, it dwarfs in comparison to the set that users can accomplish with existing mobile apps.

Therefore, virtual assistant platforms have started leveraging apps to accomplish a wider set of tasks. Platform-specific mechanisms such as *Android App Actions* [13] and *Siri Shortcuts* [21] allow developers to programmatically expose the capabilities of installed apps to virtual assistants, in effect augmenting an assistant's skill set. End users can also endow virtual assistants with new skills by creating simple logic flows using platforms like IFTTT [31], Workflow [54], and Shortcuts [52]. All of these options, however, require *manual* development effort.

This paper presents SAVANT[1], a system that automatically generates task shortcuts for virtual assistants by mapping user tasks to relevant UI screens in apps (Figure 1). SAVANT then leverages these shortcuts to launch apps at task-relevant screens. While users can directly interact with SAVANT through voice- and text-based input, virtual assistants can also interface with the system to automatically discover and launch UI states within installed apps, which can be used to perform tasks that the assistants do not currently handle.

Given a natural language task description (e.g., "send money to Joe"), SAVANT leverages text and semantic information contained within UIs to identify relevant screens, and intent modeling to parse and map entities (e.g., "Joe") to required UI inputs. A task often specifies an action such as "send" or "shop," and an object of that

[1] **S**hortcuts in **A**pps for **V**irtual **A**ssistant **N**ew **T**asks

action such as "money" or "clothing." UI elements contained within a screen such as *icons* and *buttons* often represent actions, while the object of these actions can be inferred from surrounding *text fields*. SAVANT combines icon, button, and text data from UI components along with an app's Google Play Store metadata to match tasks to UI screens and produce *task shortcuts*.

A task shortcut contains a programmatic reference to an Android app's UI screen (i.e., *activity* name), which SAVANT can leverage to launch apps at task-relevant screens. Additionally, a task shortcut contains a previously recorded interaction trace that captures a path from an app's home screen to the shortcut's UI screen. If SAVANT is unable to directly launch a shortcut's UI screen, it attempts to navigate to the screen by replaying the interaction trace.

In essence, SAVANT is an unsupervised search system that finds the best-matching app screen for a given task by leveraging a large corpus of unlabeled interaction traces. SAVANT not only identifies task-relevant apps, but also finds screens within apps that can serve as a starting point for the task. In fact, SAVANT complements existing programming-by-demonstration (PBD) systems by providing them with starting points for task demonstrations they can generalize from.

This paper demonstrates that SAVANT effectively identifies UI screens that are relevant to user tasks. Through a 24-person study and iterative coding, we identified 20 sets of common smartphone tasks. We then bootstrapped SAVANT using the *Rico* dataset, comprising UI screens and interaction traces from 9.3k Android applications [11]. We queried SAVANT with representative tasks from each of the 20 sets, and asked three participants to evaluate the top-3 results; SAVANT achieved 70.1% average precision.

2 BACKGROUND & MOTIVATION

Virtual assistants are capable of automatically performing several everyday tasks with simple voice commands. Leading virtual assistant technologies are now included with all new smartphones — Google Assistant for Android and Siri for iOS — enabling many task automation opportunities that leverage installed apps. However, virtual assistants are limited by the functionalities the app developers make available for them. If app developers do not make an extra effort to externally expose specific tasks in their apps, then virtual assistants are left in the dark.

Even when developers extend these efforts, there are still limitations for both Android and iOS platforms. In Android terminology, an *intent* is a description of an operation to be performed [16]; therefore built-in intents are built-in operations that Google Assistant can perform. Google Assistant only supports a limited number of built-in intents that apps can interface with. While iOS app developers can encode custom intents to interface with Siri, the end-user still has to manually configure Siri to allow the per-app automations.

Consider the PayPal app, which allows its users to send money to other users. PayPal's Android developers can use *App Actions* [13] to notify Google Assistant that it can use PayPal whenever a user wants to send money. To use an *App Action*, the developers need to add a file (`actions.xml`) to their app which declares to Google Assistant that PayPal is able to handle the built-in intent for "create money transfer" [14]. Although the list of built-in intents is growing, it is not yet possible to enable Google Assistant to automatically use apps for custom tasks.

To enable Siri to automatically send money through PayPal on iOS, the process is similar. The iOS developers need to add an `Intent Definition File` to their app that declares the tasks the app supports (i.e., iOS *intents*). As opposed to Google Assistant, Siri is compatible with custom intents on top of system intents. Although iOS provides more flexibility, developers also need to define a mechanism that *donates* the shortcut to Siri every time the user sends money using PayPal [20]. Siri will start using the shortcut automatically only after it is donated by the app. Developers can also expose app capabilities to Siri via *Suggested Shortcuts* [22], which requires end-user configuration for task automation.

In contrast with these approaches that require additional development effort, SAVANT leverages the information within app UIs to match screens with tasks and automatically generate shortcuts that launch task-relevant screens within apps. SAVANT is motivated by the observation that keywords in task descriptions are often semantically related to UI components. Therefore, it leverages data collected from *interaction mining* [12] — sequences of app screens augmented with render-time properties (i.e., view hierarchies) — to find semantically related UI screens for a given task description. Developers do not have to manually instrument an app's code to externally expose its capabilities; SAVANT can automatically discover an app's capabilities based on a set of interactions traces recorded for the app. For example, SAVANT can automatically identify the UI state in PayPal where a user can send money and generate a shortcut for that task. Then, SAVANT can use the shortcut to automatically navigate the virtual assistant (or the end user) to this UI screen.

3 RELATED WORK

SAVANT generates task shortcuts that allow external systems — such as virtual assistants — to automatically navigate users to task-relevant app screens. The shortcuts it generates depend on the repository of interaction traces it searches over. For example, end-users can configure SAVANT to search over interaction traces for any app that is available on the Play Store, behaving more like an app recommendation system. If a generated shortcut points to an app that is not installed on the device, other tools can integrate with SAVANT to download it from the app store and automatically navigate users directly to the task-relevant screen. On the other hand, end-users can limit SAVANT's search to interaction traces apps already installed on their devices. In this configuration, the generated shortcuts behave similarly to deep links, enabling direct navigation to a task-relevant UI screen in an app. It is important to note that in both of these scenarios, SAVANT does not aspire to be a task automation system, nor does it replicate their functionality.

3.1 Virtual Assistants

Given the increasing popularity of virtual assistants [44], recent work has focused on improving their user experience [10, 49] and augmenting their capabilities [24, 58]. Virtual assistants can already provide users more information about the content on a screen [28, 46]. Moreover, systems like *JustSpeak* enable virtual assistants to access developer provided labels for UI elements [58]. However,

none of these systems evaluate a screen with respect to the tasks it can help users complete. SAVANT leverages semantic annotation techniques [41] to match UI screens and their elements to user tasks.

3.2 App and Feature Recommendation

Personalized app discovery systems can recommend *new* apps to install based on usage patterns [56], context (e.g., location, time) [32], and privacy preferences [40, 59]. Unlike SAVANT, these systems do not directly link user tasks to apps. SAVANT semantically analyzes UI elements on app screens to provide task-specific app recommendations.

Existing approaches predict and recommend *installed* apps to launch based on spatiotemporal [4], sensory [51], and usage pattern [53] data. These system enable users to quickly start relevant apps; however, unlike SAVANT, these approaches lack mechanisms to directly launch apps at task-relevant screens.

Translating natural language task descriptions to application capabilities has also been explored through mapping user tasks to application commands [1, 25], and providing conversation-relevant shortcuts for messaging apps from external apps [7]. These systems only match tasks to features of *specific* apps, whereas SAVANT is able to work with *any* app provided previously recorded interaction traces.

3.3 Mobile Deep Links

Deep linking conventionally refers to providing a uniform resource identifier (URI) that grants direct access to a specific page within a website rather than its home page. In the context of mobile applications, these URIs point to specific UI screens within apps instead of pages within websites. The importance of deep links in mobile apps has been recognized recently, especially with the rise of novel interaction paradigms like virtual assistants. Virtual assistants rely on third-party apps to fulfill many queries from the user, but currently, the onus is on app developers to manually expose and provide deep links to the actions supported by their apps.

The Android platform allows app developers to manually create deep links for their apps using intent filters, which allow developers to specify the URI and/or the MIME type of the data that their app is able to handle [15]. This type of linking targets navigation to different content inside apps. To enable a wider set of actions for use with virtual assistants, Google introduced *App Actions*; however, developers have to manually add support for these predefined actions [13].

Researchers have explored different approaches for helping developers add deep links into their apps. Ma et al. proposed RESTful-style app models to improve service discovery and content search [43]; Azim et al. implemented *uLink*, a developer library which allows generation of user-defined deep links through instrumentation [3]; and Ma et al. proposed *Aladdin*, an approach using static and dynamic program analysis to generate deep link APIs for apps which developers can choose to expose from their own apps [42].

All of these existing methods still require developer effort to connect virtual assistants to functionality within apps. SAVANT provides automatic task shortcuts that enable navigation to a task-relevant screen within an app. Although deep links can contain more information than SAVANT's task shortcuts to automate user actions, they still impose a burden on the app developers as indicated by their low adoption rate [29, 42]. SAVANT is a zero-integration alternative for deep links.

3.4 Task Automation

Several approaches use programming-by-demonstration (PBD) techniques to automate smartphone tasks. Systems such as *PUMICE* [37], *KITE* [38], *SUGILITE* [36], and *VASTA* [50] are akin to supervised learning: a user demonstrates a particular task in a particular app, and the system learns how to generalize the interaction and replay it with different parameters.

In contrast, SAVANT is an unsupervised search system. Users are not asked to demonstrate or label tasks: instead, the system runs on a large, unlabeled corpus of interaction traces from many users. SAVANT does not generalize or automate: it merely finds the best-matching app screen for a given task. We believe task shortcuts will integrate well with existing task automation tools, as they provide means for automatically launching an app at a task-relevant screen and potentially cut down considerable demonstration effort.

Recent research has explored opportunities to interface with task automation systems. *X-Droid* uses task automation with existing app components to build functional app prototypes [33]. *PUMA* explores UI automation techniques to aid large-scale dynamic analysis [27]. *Humanoid* leverages interaction traces for test input generation for automated UI testing [39]. As opposed to SAVANT, these methods leverage task automation techniques to achieve their own goals; existing task automation tools can interface with SAVANT to improve their user experience.

3.5 Intent Modeling

Intent matching is the process by which words or expressions are automatically parsed and mapped to specific desired tasks or outcomes. For example, the expression "send money to Joe" could be matched to an intent such as *send money*, and the entity associated with that intent — "Joe" — can be extracted.

Prior task automation systems utilize intent matching to interface with third-party applications [36, 37], or build conversational bots [38]. In systems such as *SUGILITE* [36] and *PUMICE* [37], intent matching works closely with the PBD component to recognize intents and entities. In contrast, SAVANT leverages *Dialogflow*, which requires only a few example phrases for each intent-entity pair to train a robust agent [9], eliminating the need for a PBD component. Moreover, SAVANT is able to maintain the flexibility of systems like *KITE* [38] through *Dialogflow*'s "automated expansion" feature for entities, which automatically recognizes new values for entities based on pattern matching.

4 THE SAVANT SYSTEM

SAVANT maps user tasks to relevant app screens, and then launches those apps directly at those screens. SAVANT's screen search system leverages data captured through *interaction mining* [12]. This data comprise screen sequences — *interaction traces* — that capture each screen's visual (i.e., screenshots) and structural (i.e., view hierarchies) render-time properties. For each screen in the set of available

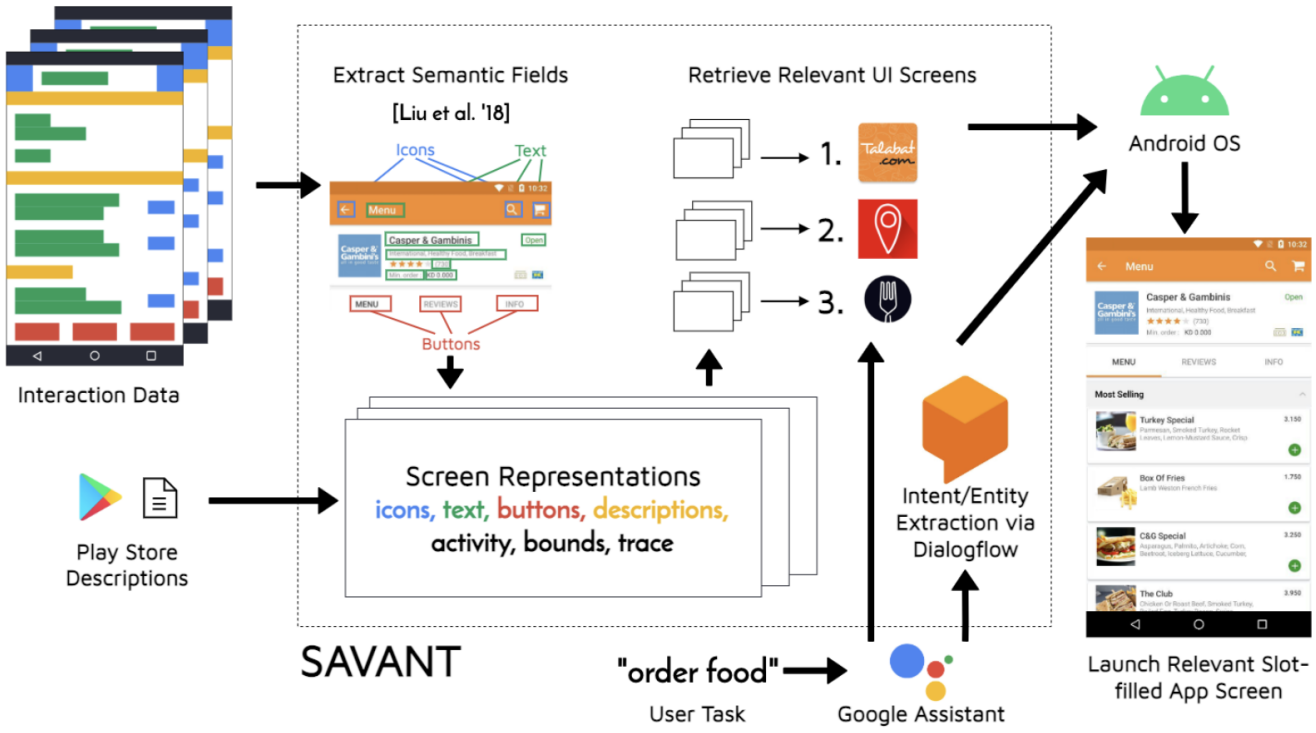


Figure 2: SAVANT matches user tasks (e.g., "order food") to app screen representations computed over semantically annotated interaction data and app descriptions from the Google Play Store. SAVANT then directs the Android OS to launch the best-matching app at the most task-relevant screen and populates the screen's UI elements with extracted entity values associated with the task intent.

interaction traces, SAVANT computes a semantic representation, which is used to find screens that best match a given user task and ultimately generate a **task shortcut**. SAVANT leverages the shortcut to launch the most task-relevant app screen and populates the screen's UI elements with extracted entity values associated with the task intent. (Figure 2).

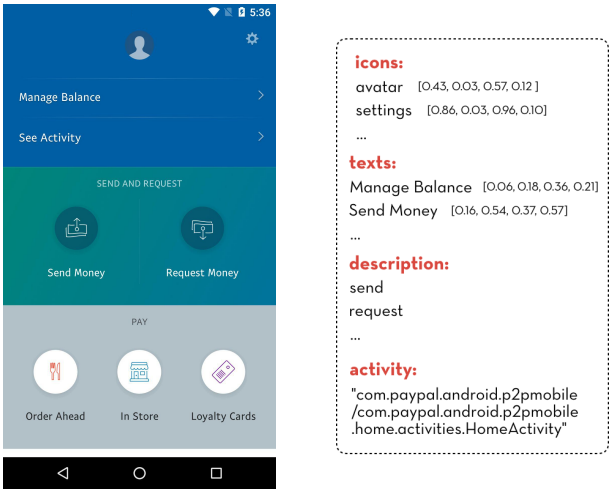
4.1 Generating a Task Shortcut

UI components contain semantic information which can be used to match them to user tasks. In order to gain access to this information, SAVANT utilizes user traces collected through *interaction mining* which captures design and interaction data while an Android app is being used [12]. These *interaction traces* comprise screen sequences illustrating users' journeys within an app. For each screen, an interaction trace encodes its screenshot, underlying structural composition and render-time properties (i.e., view hierarchy), and the user interactions performed on the screen resulting in transitions to other screens within the app (i.e., gestures).

Leveraging data contained within view hierarchies, SAVANT computes screen representations that are used to find the best matching screens for a given user task and generate task shortcuts (Figure 3). These screen representations encode semantic information about UI elements such as *icons* (e.g., Avatar, Settings, Arrow Forward), *buttons* (e.g., Send, Finish, Checkout), and the *text fields*. Following Liu et al.'s approach, we compute semantic annotations over all UI elements on screen [41]. Semantic annotations for *icons* and *buttons* are likely to match the main action or the verb in a query (e.g., "send"). On the other hand, the object of a query (e.g., "money") is more likely to be present within text fields on a screen.

Some of the text within these components occasionally lead to false matches with search queries. For example, nearly every app mentions email in multiple locations, so a task query such as "check email" matches many screens that are not from email clients. We found that including Google Play app descriptions while performing search helps mitigate this problem and provide better matches. Therefore, screen representations also include Google Play app descriptions, which have been processed to remove common stop words.

Finally, SAVANT's screen representations include programmatic references that can be used to launch an app at a specific UI state. In addition to encoding the structural composition and render-time properties of a UI, a view hierarchy contains the name of the screen's Android *activity* — essentially "an app component that provides a screen with which users can interact in order to do something" [17]. The *activity* name can be used to directly launch the app at the corresponding screen. In case a screen cannot be launched directly because it requires input parameters, SAVANT also stores a pointer to the interaction trace leading up to the screen.



(a) (b)

Figure 3: Home screen for the PayPal app (a) and its computed screen representation (b), which encodes semantic information — `icons`, `texts`, and app `description` fields — and a programmatic reference for launching PayPal at this home screen — the `activity` name.

Using an off-the-shelf cloud search service, SAVANT matches task queries to the semantic information contained within the screen representations via keyword search. SAVANT preprocesses task queries by performing basic stemming and removing common stop words that do not add value to the description. We configured the search with a set of synonyms for common actions found in mobile apps, derived from UX concepts found in Liu et al. [41], so that task queries about "chat" will also match screens containing "message" and "comment".

A SAVANT search result contains a set of screen representations with their corresponding relevance scores indicating how well the screens matched the query. SAVANT aggregates these results by app and assigns an overall score to each app equal to a weighted sum of the individual screen scores. We boost apps with more matching screens since these apps are more likely to be capable of meeting the needs of the user. Finally, SAVANT returns the top screen representation from the highest scoring app, which is called a **task shortcut**.

4.2 Using a Task Shortcut

A task shortcut contains sufficient information to launch an app at a task-relevant screen. To demonstrate how task shortcuts can be used, we encapsulated SAVANT within a *system-level* Android app. Android apps can restrict the *activities* that can be started externally from other apps; but there is a special permission that is available to system-level apps, `START_ANY_ACTIVITY`, which allow them to bypass any restriction while starting an *activity*.

Android apps use *intents*—messaging objects to request an action from another app component [16]—to launch new *activities*. When an *intent* contains information on which component will handle the action, it is considered to be an *explicit intent*; these are typically

```
1  val intent = Intent()
2  intent.component = componentName
3  intent.addFlags(FLAG_ACTIVITY_NEW_TASK)
4  context.startActivity(intent)
```

Figure 4: Kotlin code required for creating an *explicit intent* to launch a new *activity*. `componentName` contains the target app and the *activity*. `FLAG_ACTIVITY_NEW_TASK` allows creating the activity as a new task on the same history stack; when the back button is tapped, the user will be navigated to the screen where this intent originated from.

used to launch new *activities* within the same app. When an *intent* declares a general action, without containing information about the component, it is considered to be an *implicit intent*; these are typically used to delegate the action to another app chosen by the user. Although *implicit intents* are suitable for launching other apps to handle tasks, they still require users to choose the app they want to use. As a result, SAVANT uses an *explicit intent* to start the *activity* specified in the `activity` field of the task shortcut.

Using the example from Figure 3, the `activity` field within the task shortcut points to an app (the part before the "`/`") and an *activity* within the app (the part after the "`/`"). Combined, these two strings form a *component name*. SAVANT uses this *component name* to create an *explicit intent* and sends it to the Android system, essentially navigating the user to the matched screen (Figure 4).

Some *activities* require user input to be launched because they require a specific internal state for those screens to be accessible. In theory, these user inputs can be provided externally to *activities* through various `putExtra` methods of *intents*. These methods allow serializable data to be included alongside *intents*. Unfortunately task shortcuts do not support this approach yet, since it requires knowledge of how an *activity* is actually implemented. Instead, when this happens, we use the `trace` field in the task shortcut and replay the interactions leading up to the component.

The interaction trace data contains gestures: user actions performed on a UI state resulting in transitions to other UI states within the same app. Given that the interaction trace contains the matched UI state, it must also contain the starting state and the set of transitions that result in the matched UI state. Starting from the initial state, if we follow these transitions by simulating the gestures on the screen, we should end up in the matched UI state.

Since SAVANT is a system-level app, it can request the permission, `INJECT_EVENTS`, which allows it to simulate gestures on the screen. SAVANT currently supports simulating two types of gestures: touch and swipe. By simulating the gestures, SAVANT replays the interactions to reach the matched screen. Although these two gestures cover most of the transitions between UI states; sometimes other gestures such as pinch-and-zoom might be required. SAVANT injects the gestures as events on to the screen with static parameters from the interaction trace and presents the step-by-step replay to users, so that they are aware of the parameter choices that lead up to the task-relevant screen. This approach requires updated interaction

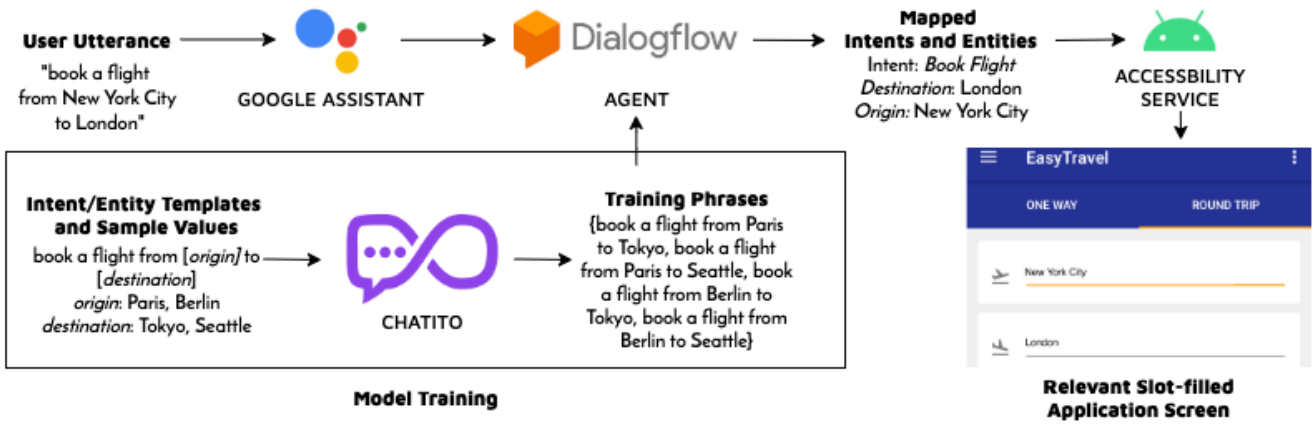


Figure 5: SAVANT's *Dialogflow* agent matches user utterances to intents and entities; the extracted entities are used to autofill the values of any corresponding UI elements on task-relevant screens. The agent is trained on phrases generated by Chatito.

traces when the UI for an app changes. When replay fails, SAVANT falls back to launching the app at its home screen.

The system-level app implementation of SAVANT supports voice- and text-based inputs directly from the user. Although SAVANT's main goal is to interface with virtual assistants and help them with tasks they do not know how to handle, end users can also directly use SAVANT through the app.

4.3 Slot-filling and Intent Modeling

A given user task — "send money to Joe" — usually comprises an intent—"send money"—and a set of associated entities, which in this case is "Joe". To make a task shortcut more useful, in addition to automatically launching a task-relevant app screen, SAVANT should autofill UI screen elements with user inputs corresponding to the entities of the user task's intent. To accomplish this goal, which is known as the slot-filling problem, we implement a *Dialogflow* agent [9]. This agent's purpose is to match user utterances to intents and entities, which can then be mapped to UI elements on the app screen (Figure 5).

The *Dialogflow* agent simultaneously uses *rule-based grammar matching* and *ML matching* methods to determine the most likely intent for a given natural language phrase. An intent in *Dialogflow* is defined as an "end-user's intention for one conversation turn". In SAVANT's' use case, the conversation consists of a single task phrase, from the user to the system, followed by SAVANT opening a specific application at a relevant UI screen. For example, the trained agent may match a user utterance to the "book flight" intent when the user expresses the desire to book a flight. While intents are analogous to verbs in sentences, entities are used to represent nouns and adjectives within natural language phrases. Building upon the "book flight" example, the agent may match part of the task phrase to a "destination" entity if the utterance contains where the user would like to fly to.

We predefined the agent's intent and entity bank based on application concepts extracted from previous work [41] and the formative study identifying common smartphone tasks (Section 5.1). This manual approach is actually extensible and efficient because the rules are defined globally across all apps and not on a per-app basis. Automatically building out these intents and entities from UI elements present in user traces would be challenging, due to the lack of order among UI elements. The elements on a UI screen are more similar to a "bag of words" representation than to a natural language utterance, and for intent and entity extraction, that kind of representation would not be useful in training an agent. Extracting intents like *book flight* from just analyzing the UI screen of an application like Expedia is difficult.

Based on the extracted UX concepts from Liu et al. [41] and the formative user study, we encoded a robust and generalizable set of intents and entities that forms the backbone of SAVANT's *Dialogflow* agent. Both of these sources of user tasks capture common intents across Android applications: *Add*, *Checkout*, *Forgot Password*, and *Terms of Service*. A subset of these concepts can be expressed as *Dialogflow* intents. For example, *Add* can be expressed as an intent, whereas *Terms of Service* cannot. This subset of concepts that are also intents will have corresponding entities; for example, the *Add* intent may refer to adding a *song* entity to a *playlist* entity.

Dialogflow agents are trained on example phrases and values for each intent and entity, respectively. For example, for an intent called *greeting*, an example training phrase might be "Hi, how are you". For an entity called *restaurant*, an example value might be "Olive Garden". To efficiently generate *Dialogflow* training phrases, we leverage *Chatito*, a commonly used dataset generation Domain Specific Language (DSL) [48]. Instead of manually writing out each training phrase, we provide *Chatito* with template intent phrases such as "book a flight from [*origin*] to [*destination*]", and a list of values for the entities (*origin* and *destination*) expressed within the template. *Chatito* then generates all possible combinations of the phrase with the different entity values filled in, significantly reducing manual effort. For a DSL comprising 25 intent templates and 20 entities, *Chatito* approximately generated 100 training phrases for each intent. We uploaded the phrases generated by *Chatito* to *Dialogflow*, and manually annotated the intent and entities in the phrases using *Dialogflow*'s web interface to train the agent.

After finding the most task-relevant screen, SAVANT sends the task phrase to the trained *Dialogflow* agent, which returns the

most likely intent and associated entities extracted from the user utterance. The entities are then compared to UI elements on the application screen, and the matching components' values are set to their corresponding entity values. Since the Android OS does not allow regular applications to access the elements of third-party applications, SAVANT uses accessibility services to access and edit elements on screens.

4.4 Implementation

We use Python scripts to preprocess the interaction mining data and compute semantic annotations. We utilize AWS CloudSearch to perform the search and ranking of screen representations. We implemented a Node.js API that handles task description stemming, querying the data in AWS CloudSearch, generating the task shortcuts, and aggregating them in apps. The system-level Android application and the accessibility service used for slot-filling are implemented in Kotlin.

5 EVALUATION

We conducted a user study to evaluate whether SAVANT's semantic search identified UI screens that are task-relevant. Although there have been many studies focusing on smartphone usage at the *app level* [5, 6, 8, 19, 34, 35, 55, 57], there are not many extensive and publicly available studies investigating smartphone usage at the *task level*. Therefore, we first conducted a formative study to identify common user tasks performed on smartphones. We then conducted a second study to evaluate SAVANT's performance on the representative task set identified by the formative study.

5.1 Identifying Common User Tasks

To the best of our knowledge, the most extensive study investigating common tasks users perform on their smartphones is the motivating study done for *SUGILITE* [36]. However, the results of this study is not publicly available and the paper only provides eight user tasks. Thus, we conducted a new 24-person formative study to identify classes of commonly performed smartphone tasks.

We leveraged methodology similar to prior work [36, 37], and asked each participant to list five tasks they would complete on a smartphone. The participants were recruited by advertising through a university web programming course mailing list. We compensated each participant with a $10 Amazon gift card. Each session lasted approximately 15 minutes.

Participants generated a total of 120 tasks. Users generally provided task descriptions that contained higher-level intents (e.g., "message friends") rather than instance-level instructions (e.g., "tell my roommate I'm running late"). SAVANT is meant to be used as an intermediary between virtual assistants and apps to provide tasks shortcuts. Virtual assistants already extract higher-level intents from instance-level instructions; therefore SAVANT would mostly need to handle higher-level intents.

We first grouped the 120 tasks with respect to the Google Play Store categories they correspond to, forming initial task sets. Inspecting these initial groups, we further categorized tasks that are very specific into their own distinct classes. For example, we split "video chat" from its initial grouping with "Communication" tasks since the remaining tasks were all for text-based communication.

TASK SET	PRECISION (%)
message friends (5), text friends (3), check email (2), communicate with people (2), chat with friends, group chat, message people, messaging, respond to emails, send messages, write email	55.6
listen to music (6), play music (3), watch videos (2), browse forums, listen to podcast, recipe videos, watch movies	88.9
set an alarm (6), set a reminder (2), set alarm (2), set alarms, set reminder	55.6
send money (3), check bank account (2), deposit a check, make a payment, make payments, mobile payments, paying people, track spending	100
check weather (3), check the weather (2), calculator, get sports updates, identify a song, search online, show tickets, take measurements, what time is it	77.8
check calendar (2), **view calendar (2),** add tasks, calendar, calendar to do list, view calendar events, create events, to do list	66.7
get directions (2), find directions, navigate, navigating to a destination, navigation, transit time	77.8
take notes (4), taking notes, write down notes	44.4
take photos (2), scan a document, take a picture, take pictures	100
post a picture, post pictures, post updates	11.1
track diet, **track workout,** view workout activity	66.7
order food (2), buy groceries	100
read books (2), view document	33.3
video chat (2)	66.7
find deals (2)	100
translate word, translate words	88.9
book a cab, make reservation	77.8
turn off lights	33.3
practice flashcards	100
play games	66.7

Figure 6: The 20 task sets identified through the formative study along with SAVANT's top-3 precision for each set. The representative tasks for each set are in bold. The parentheticals indicate the task's frequency in the study results.

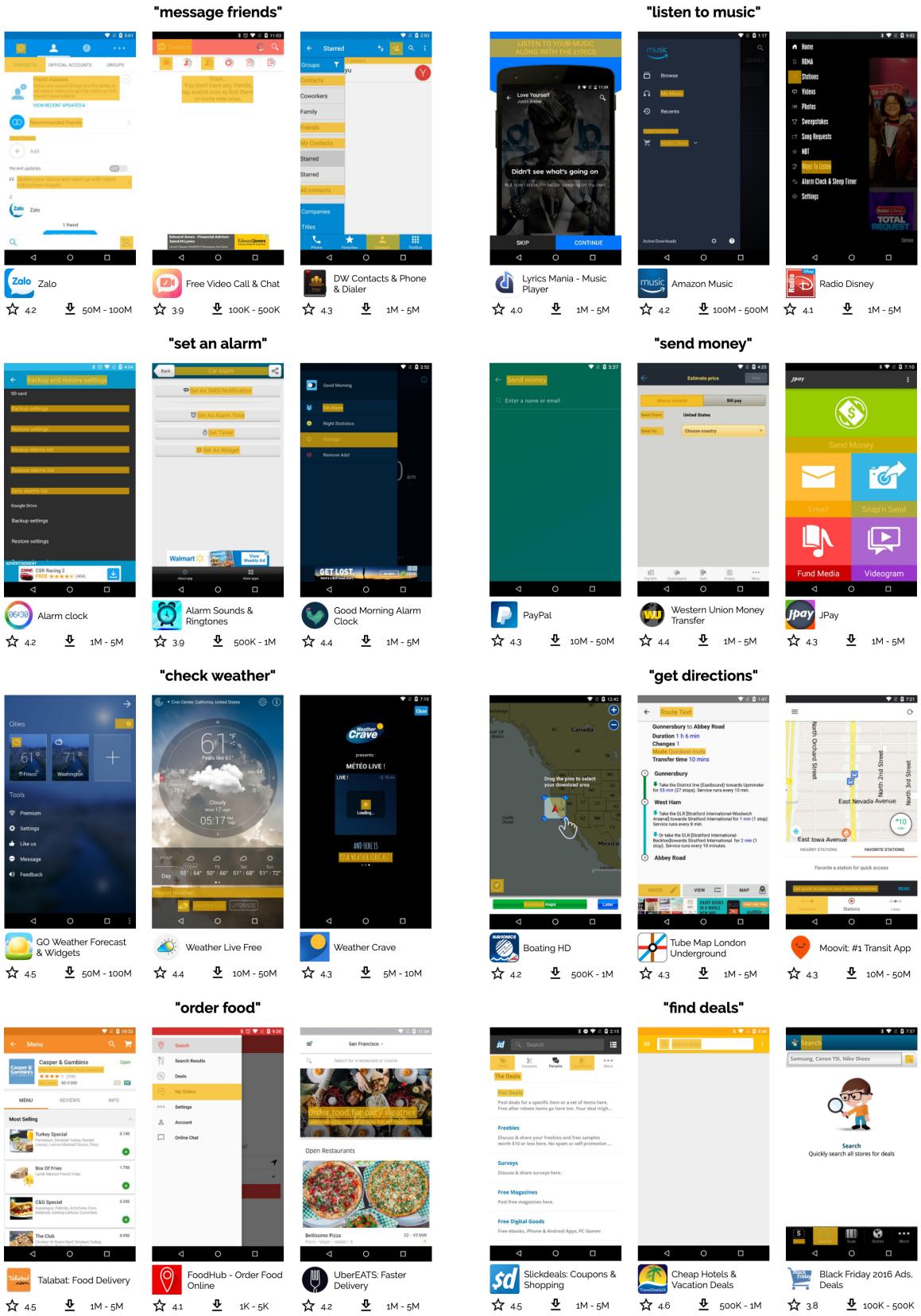


Figure 7: SAVANT's screen matching results for representative tasks, in descending screen relevance order from left to right. Elements matching the tasks are highlighted for additional clarity.

We also merged together some task sets which were thematically related (e.g., "Tools" and "Weather"). This process resulted in 20 unique sets of tasks. The most frequently occurring task in a set became the representative task for that set. Ties were decided by picking the task phrase that had the most common words with the rest of the set (Figure 6).

5.2 Measuring Task Relevance Precision

We recruited three participants to evaluate the quality of shortcuts suggested by SAVANT. These participants were recruited from the same mailing list as the formative study (Section 5.1) and were also compensated with a $10 Amazon gift card.

We first bootstrapped SAVANT with the *Rico* dataset, comprising semantic and textual data for 66k UI screens from 9.3k Android apps [11, 41]. We computed the top-3 task shortcuts for each representative task identified in the formative study. We chose to display only the top-3 results since three was the maximum number of shortcuts generated for some of the representative tasks. This is also consistent with how virtual assistants behave when there are multiple apps that can help with a given task.

For each task, we showed participants the top-3 results returned by SAVANT, and asked them to evaluate whether or not each screen was task-relevant. We only explained that a relevant task shortcut navigates users to a UI screen that helps them complete the specified task. Similar methodology was employed in prior research by Chen et. al. where participants were asked to identify the "top-3 relevant shortcuts for apps" from the context of a text message [7].

Task success rate is not a relevant metric for SAVANT, since it is not a task automation system. We also do not measure *recall* mainly because it would require exhaustive labeling over 66k UI screens for each task which is infeasible. Recall is also not particularly helpful for the general use case: providing users with top-k task shortcuts would be more than enough to help them accomplish their tasks, which is the main purpose of SAVANT.

We computed top-3 precision for each of the 20 task sets (Figure 6). The average precision over all task sets was 70.1%. SAVANT had 100% top-3 precision for five tasks: "send money", "take photos", "order food", "find deals" and "practice flashcards". For 13 out of the 20 task sets (65%), SAVANT provided at least two relevant task shortcuts, and the average top-3 precision was greater than 66% (Figure 7).

Still, SAVANT performed poorly for some task sets. For "take notes" (44.4%), SAVANT returned shortcuts from domain-specific apps such as myPill® Birth Control Reminder and Blue Letter Bible that allow taking notes as side features. SAVANT also suggested a shortcut from the FlightView Free Flight Tracker app since it treated the verb "take" as a synonym for "flight", "take off". "Post a picture" produced the worst results (11.1%) since SAVANT only found shortcuts that would either allow users to make a "post" on a forum or take a "picture". Two of the results for "read books" (33.3%) were from Hotels.com and SpiceJet apps, which highlight the limitation that SAVANT is is currently unable to distinguish between verbs and nouns. The only useful shortcut for this task was from the Free Books app. Finally, for "turn off lights" (33.3%), SAVANT suggests shortcuts from apps for configuring screen lighting for the night (Night Light and Blue Light Filter) while the participants were expecting apps for managing smart home devices. For this task, the only shortcut the participants found relevant was from the Color Lights Flashing app.

6 DESIGN CONSIDERATIONS FOR REAL-WORLD DEPLOYMENT

This paper presents a new approach to augmenting virtual assistant capabilities that leverages a repository of previously recorded app interaction traces. While bootstrapping SAVANT with a public interaction data repository like *Rico* [11] was a convenient strategy to demonstrate the viability of this approach, SAVANT would need to address a few key challenges to function robustly in the real-world. A real-world deployment would require strategies for maintaining a repository of up-to-date app traces, addressing potential security and privacy concerns, and expanding to platforms beyond the Android ecosystem.

6.1 Sustainably Sourcing Traces

In the future, relying only on paid crowdworkers for sourcing high-quality, up-to-date app traces does not scale. In addition to being a costly solution, it would be challenging to determine which apps to sample traces from beyond just the popular ones. Therefore, we envision a future ecosystem where app interaction traces could be sourced in two additional ways: having app developers provide sample interaction traces and directly mining interactions from user devices.

App developers could include sample interaction traces along with their apps when they update it. While the first set of sample traces would need to cover all of the UI screens within the app, the subsequent traces would only need to refresh new and updated UI screens. Moreover, app developers can themselves source interaction traces from crowdworkers or users, tasking them with re-crawling potentially out-of-date flows. App developers would be incentivized to contribute interaction traces to a service like SAVANT so that their app would be automatically discovered, downloaded, and ultimately used by virtual assistants when it is relevant to a user's task.

In the future, end users might also be incentivized to share their own app interaction traces. An on-device interaction mining app could record users' interactions on their smartphone as they engage with apps. Systems like SAVANT could leverage these interaction traces to provide more personalized task shortcuts for apps that users already have on their phones. When a new app is downloaded, these system could bootstrap task shortcuts with traces from a public interaction data repository until users generate their own personal traces from engaging with the app. End users could opt in to contribute their own personal interaction traces back to this public interaction trace repository.

Combining interaction data from both public and personal trace pools could significantly improve SAVANT's user experience. If a user already has a task-relevant app installed and personal interaction traces recorded for this app, SAVANT would produce results that are more familiar to its users. When a user does not have a task-relevant app installed, SAVANT can use the public repository of traces to suggest an app to bedownloaded and provide the shortcut

for the task. If a user has the task-relevant app installed, but has not used the app enough to generate personalized interaction traces, SAVANT can leverage traces from the public repository to generate the task shortcut.

6.2 Addressing Security and Privacy Concerns

Leveraging a system-level app for UI automation poses some security concerns. System-level apps can request special permissions that are not available to normal apps, like installing other applications, being able to reboot the device, and setting the system time. SAVANT uses two special permissions: one that allows it to externally launch activities within other apps (`START_ANY_ACTIVITY`) and one to simulate interaction gestures on the screen (`INJECT_EVENTS`). Both permissions are only used with interaction data to automatically navigate users to task-relevant UI screens. This means, for a security problem to occur, the trace SAVANT uses for a task shortcut needs to contain malicious UI screens and interactions. Google Play Store already has measures to detect and prevent the installation of apps that might contain such UI screens [18]. It is also unlikely for SAVANT to match a UI screen from a malicious trace with a specific user task.

In terms of privacy, as with any system that leverages user interaction data, SAVANT faces the risk of exposing personally identifiable information (PII) of its users. Interaction mining techniques capture design and interaction data while an app is being used; therefore, interaction traces undoubtedly contain sensitive data. The future SAVANT ecosystem will have to handle user PII in a way that encourages the contribution of interaction traces while strictly protecting user data.

The most basic solution to this problem is to create and provide sock puppet accounts that trace contributors can use to explore apps. During app development, similar accounts are already used to perform quality assurance. App developers can use those accounts to provide sample interaction traces along with their apps. Limited versions of such accounts can also be provided for crowdworkers for re-crawling out-of-date traces. This solution, however, does not address the case for a potential on-device mining system that records interactions from real users.

If SAVANT is to accommodate recording traces as they naturally happen while users interact with apps on their smartphones, one potential solution would be to process UI screens when they are first recorded to automatically detect and remove PII. There are existing systems that can automatically detect conventional PII like social security numbers (SSN), email addresses, phone numbers, etc. through dynamic [23] and static [2] program analysis, natural language processing [45], and text analysis [26, 30]. Therefore, using interaction data with app binaries would enable identifying and removing components that contain conventional PII.

However, more recent regulations like GDPR, define *personal data* as "any information which is related to an identified or identifiable natural person" [47]. Further research is required to develop user-centered privacy models that identify sensitive information beyond what is conventionally considered PII. In the future, on-device mining systems could expose interfaces where end users can redact sensitive information from their personal traces, especially if they are being included in the global repository.

6.3 Expanding to Other Platforms

While the general approach of leveraging previously recorded app interaction traces to augment virtual assistant capabilities is platform agnostic, SAVANT currently only interfaces with Android apps and Google Assistant. We chose to develop SAVANT within the Android ecosystem because it exposes the system-level capabilities that were required: mining interaction traces, and programmatically launching UI screens and performing interactions.

Although iOS allows for some types of gesture automation, it still lacks a public interface for programmatically launching UI states. While it might still be possible to implement SAVANT as a system-level iOS application, it would require knowledge of mechanisms/APIs within iOS that are not public.

In the future, SAVANT could even interface with apps deployed on smart devices like televisions, speakers, and other appliances. Mining and programmatically interacting with non-touchscreen UIs in such apps could extend this approach beyond smartphones.

ACKNOWLEDGMENTS

We thank the reviewers for their helpful comments and suggestions. This work was supported in part by a research donation from Google Faculty Research Award and NSF Grant IIS-1750563.

REFERENCES

[1] Eytan Adar, Mira Dontcheva, and Gierad Laput. 2014. CommandSpace: modeling the relationships between tasks, descriptions and features. In *Proc. UIST*. ACM, 167–176.
[2] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices* 49, 6 (2014), 259–269.
[3] Tanzirul Azim, Oriana Riva, and Suman Nath. 2016. uLink: Enabling user-defined deep linking to app content. In *Proc. MobiSys*. ACM, 305–318.
[4] Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. 2015. Predicting the next app that you are going to use. In *Proc. WSDM*. ACM, 285–294.
[5] 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 *Proc. MobileHCI*. ACM, 47–56.
[6] Juan Pablo Carrascal and Karen Church. 2015. An in-situ study of mobile app & mobile search interactions. In *Proc. CHI*. ACM, 2739–2748.
[7] Fanglin Chen, Kewei Xia, Karan Dhabalia, and Jason I Hong. 2019. MessageOnTap: A Suggestive Interface to Facilitate Messaging-related Tasks. In *Proc. CHI*. ACM, 575.
[8] Karen Church, Denzil Ferreira, Nikola Banovic, and Kent Lyons. 2015. Understanding the challenges of mobile phone usage data. In *Proc. MobileHCI*. ACM, 504–514.
[9] Google Cloud. 2021. Dialogflow. https://cloud.google.com/dialogflow
[10] 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 *Proc. MobileHCI*. ACM, 43.
[11] 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 *Proc. UIST*. ACM, 845–854.
[12] Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction mining mobile apps. In *Proc. UIST*. ACM, 767–776.
[13] Android Developers. 2021. App Actions. https://developers.google.com/actions/appactions
[14] Android Developers. 2021. Built-in intents. https://developers.google.com/actions/reference/built-in-intents/
[15] Android Developers. 2021. Handling Android App Links. https://developer.android.com/training/app-links
[16] Android Developers. 2021. Intents and Intent Filters. https://developer.android.com/guide/components/intents-filters
[17] Android Developers. 2021. Introduction to Activities. https://developer.android.com/guide/components/activities/intro-activities
[18] Google Developers. 2021. Play Protect. https://developers.google.com/android/play-protect

[19] Trinh Minh Tri Do, Jan Blom, and Daniel Gatica-Perez. 2011. Smartphone usage in the wild: a large-scale analysis of applications and context. In *Proc. ICMI*. ACM, 353–360.

[20] Apple Developer Documentation. 2021. Donating Shortcuts. https://developer.apple.com/documentation/sirikit/donating_shortcuts

[21] Apple Developer Documentation. 2021. Siri Shortcuts. https://developer.apple.com/design/human-interface-guidelines/sirikit/overview/siri-shortcuts/

[22] Apple Developer Documentation. 2021. Suggesting Shortcuts to Users. https://developer.apple.com/documentation/sirikit/shortcut_management/suggesting_shortcuts_to_users

[23] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems* 32, 2 (2014), 1–29.

[24] Ethan Fast, Binbin Chen, Julia Mendelsohn, Jonathan Bassen, and Michael S Bernstein. 2018. Iris: A conversational agent for complex tasks. In *Proc. CHI*. ACM, 473.

[25] Adam Fourney, Richard Mann, and Michael Terry. 2011. Query-feature graphs: bridging user vocabulary and system functionality. In *Proc. UIST*. ACM, 207–216.

[26] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking app behavior against app descriptions. In *Proc. ICSE*. 1025–1035.

[27] Shuai Hao, Bin Liu, Suman Nath, William GJ Halfond, and Ramesh Govindan. 2014. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps. In *Proc. MobiSys*. ACM, 204–217.

[28] Google Assistant Help. 2019. Find info about what's on your screen. https://support.google.com/assistant/answer/7393909

[29] Ziniu Hu, Yun Ma, Qiaozhu Mei, and Jian Tang. 2017. Roaming across the castle tunnels: An empirical study of inter-app navigation behaviors of Android users. *arXiv preprint arXiv:1706.08274* (2017).

[30] Jianjun Huang, Xiangyu Zhang, Lin Tan, Peng Wang, and Bin Liang. 2014. Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction. In *Proc. ICSE*. 1036–1046.

[31] IFTTT. 2021. Every thing works better together - IFTTT. https://ifttt.com/

[32] Alexandros Karatzoglou, Linas Baltrunas, Karen Church, and Matthias Böhmer. 2012. Climbing the app wall: enabling mobile app discovery through context-aware recommendations. In *Proc. CIKM*. ACM, 2527–2530.

[33] Donghwi Kim, Sooyoung Park, Jihoon Ko, Steven Y Ko, and Sung-Ju Lee. 2019. X-Droid: A Quick and Easy Android Prototyping Framework with a Single-App Illusion. In *Proc. UIST*. 95–108.

[34] Uichin Lee, Joonwon Lee, Minsam Ko, Changhun Lee, Yuhwan Kim, Subin Yang, Koji Yatani, Gahgene Gweon, Kyong-Mee Chung, and Junehwa Song. 2014. Hooked on smartphones: an exploratory study on smartphone overuse among college students. In *Proc. CHI*. ACM, 2327–2336.

[35] Huoran Li, Xuan Lu, Xuanzhe Liu, Tao Xie, Kaigui Bian, Felix Xiaozhu Lin, Qiaozhu Mei, and Feng Feng. 2015. Characterizing smartphone usage patterns from millions of android users. In *Proc. IMC*. ACM, 459–472.

[36] Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. 2017. SUGILITE: creating multimodal smartphone automation by demonstration. In *Proc. CHI*. ACM, 6038–6049.

[37] Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M Mitchell, and Brad A Myers. 2019. PUMICE: A Multi-Modal Agent that Learns Concepts and Conditionals from Natural Language and Demonstrations. In *Proc. UIST*. 577–589.

[38] Toby Jia-Jun Li and Oriana Riva. 2018. KITE: Building conversational bots from mobile apps. In *Proc. MobiSys*. ACM, 96–109.

[39] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2019. Humanoid: A deep learning-based approach to automated black-box android app testing. In *Proc. ASE*. IEEE, 1070–1073.

[40] Bin Liu, Deguang Kong, Lei Cen, Neil Zhenqiang Gong, Hongxia Jin, and Hui Xiong. 2015. Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. In *Proc. WSDM*. ACM, 315–324.

[41] Thomas F Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning design semantics for mobile apps. In *Proc. UIST*. ACM, 569–579.

[42] Yun Ma, Ziniu Hu, Yunxin Liu, Tao Xie, and Xuanzhe Liu. 2018. Aladdin: Automating Release of Deep-Link APIs on Android. In *Proc. WWW*. International World Wide Web Conferences Steering Committee, 1469–1478.

[43] Yun Ma, Xuanzhe Liu, Meihua Yu, Yunxin Liu, Qiaozhu Mei, and Feng Feng. 2015. Mash droid: An approach to mobile-oriented dynamic services discovery and composition by in-app search. In *Proc. ICWS*. IEEE, 725–730.

[44] Microsoft. 2019. 2019 Voice report: Consumer adoption of voice technology and digital assistants. https://about.ads.microsoft.com/en-us/insights/2019-voice-report

[45] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. 2015. Uipicker: User-input privacy identification in mobile applications. In *Proc. {USENIX} Security Symposium*. 993–1008.

[46] Jordan Novett. 2019. Microsoft beats Google to the punch: Bing for Android update does what Now on Tap will do. https://venturebeat.com/2015/08/20/microsoft-beats-google-to-the-punch-bing-for-android-update-does-what-now-on-tap-will-do/

[47] European Parliament and Council of the European Union. 2018. General Data Protection Regulation. https://eur-lex.europa.eu/eli/reg/2016/679/oj

[48] Rodrigo Pimentel. 2021. Chatito. https://github.com/rodrigopivi/Chatito

[49] Xin Rong, Adam Fourney, Robin N Brewer, Meredith Ringel Morris, and Paul N Bennett. 2017. Managing uncertainty in time expressions for virtual assistants. In *Proc. CHI*. ACM, 568–579.

[50] 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 *Proc. IUI*. 22–32.

[51] Choonsung Shin, Jin-Hyuk Hong, and Anind K Dey. 2012. Understanding and prediction of mobile application usage for smart phones. In *Proc. Ubicomp*. ACM, 173–182.

[52] Apple Support. 2021. Use Siri Shortcuts. https://support.apple.com/en-us/HT209055

[53] Chang Tan, Qi Liu, Enhong Chen, and Hui Xiong. 2012. Prediction for mobile application usage patterns. In *Nokia MDC Workshop*, Vol. 12.

[54] Workflow. 2021. Workflow - Powerful automation made simple. https://workflow.is/

[55] Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. 2011. Identifying diverse usage behaviors of smartphone apps. In *Proc. IMC*. ACM, 329–344.

[56] Bo Yan and Guanling Chen. 2011. AppJoy: personalized mobile application discovery. In *Proc. MobiSys*. ACM, 113–126.

[57] Sha Zhao, Julian Ramos, Jianrong Tao, Ziwen Jiang, Shijian Li, Zhaohui Wu, Gang Pan, and Anind K Dey. 2016. Discovering different kinds of smartphone users through their application usage behaviors. In *Proc. UbiComp*. ACM, 498–509.

[58] Yu Zhong, TV Raman, Casey Burkhardt, Fadi Biadsy, and Jeffrey P Bigham. 2014. JustSpeak: enabling universal voice control on Android. In *Proc. W4A*. 1–4.

[59] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. 2014. Mobile app recommendations with security and privacy awareness. In *Proc. KDD*. ACM, 951–960.