Discovering Natural Language Commands in Multimodal Interfaces

Arjun Srinivasan*
Georgia Inst. of Technology
Atlanta, GA
arjun010@gatech.edu

Mira Dontcheva
Adobe Research
Seattle, WA
mirad@adobe.com

Eytan Adar
University of Michigan
Ann Arbor, MI
eadar@umich.edu

Seth Walker
Adobe Research
San Francisco, CA
sewalker@adobe.com

ABSTRACT

Discovering what to say and how to say it remains a challenge for users of multimodal interfaces supporting speech input. Users end up "guessing" commands that a system might support, often leading to interpretation errors and frustration. One solution to this problem is to display contextually relevant command examples as users interact with a system. The challenge, however, is deciding when, how, and which examples to recommend. In this work, we describe an approach for generating and ranking natural language command examples in multimodal interfaces. We demonstrate the approach using a prototype touch- and speech-based image editing tool. We experiment with augmentations of the UI to understand when and how to present command examples. Through an online user study, we evaluate these alternatives and find that in-situ command suggestions promote discovery and encourage the use of speech input.

CCS CONCEPTS

• **Human-centered computing → Natural language interfaces**; *Human computer interaction (HCI)*;

KEYWORDS

Natural language interaction, Multimodal Interfaces, Discoverability, Adaptive Interfaces, Photo Editing

ACM Reference Format:
Arjun Srinivasan, Mira Dontcheva, Eytan Adar, and Seth Walker. 2019. Discovering Natural Language Commands in Multimodal Interfaces. In *24th International Conference on Intelligent User Interfaces (IUI '19), March 17–20, 2019, Marina del Rey, CA, USA,* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.1145/3301275.3302292

1 INTRODUCTION

Discovering *what* spoken commands a system will understand, and *how* commands should be phrased, remains a long-standing challenge for users of natural language interfaces (NLIs). Improvements in speech-to-text engines and prevalence of commercial speech interfaces as part of speech-only and multimodal solutions have introduced more end-users to this modality [5, 9, 27, 32, 35, 39]. However, the 'invisible' nature of speech, relative to other graphical user interface (GUI) elements, makes it particularly challenging for users to learn and adopt. Discoverability, in this context, entails: (1) *awareness*—making users aware of the operations that can be performed using speech; and (2) *understanding*—educating users on how requests should be phrased so the system can interpret them correctly. Lack of support for these discovery-oriented design goals often results in users having to guess or 'hunt-and-peck' for supported commands and phrasings [31, 47]. However, because guesses are more likely to be misinterpreted and cause increased errors, people may be discouraged from using speech input altogether.

Multimodal interfaces supporting speech+touch or speech+mouse input offer an advantage over speech-only interfaces. Because these modalities have complementary strengths [10], touch can help people use speech more effectively and vice-versa. For example, in a multimodal document reader, a speech-only interface may make it hard to ask for the pronunciation of a word. The end-user would need to guess the pronunciation of the same word they want the system to pronounce. Alternatively, with a speech+touch interface, a user can *point* to a word and ask for its pronunciation. Conversely, speech can help touch-only or mouse-based interfaces too. For instance, consider the case of a drawing tool where a user wants to add multiple instances of a red rectangle. The user first needs to learn how to add a shape and style it using the GUI, and then repeat this operation multiple times. The speech-driven alternative would be for the end-user to simply point to the canvas and say, "*add three red rectangles here*." As applications begin to support more intelligence (e.g., entity recognition in images), the opportunity for multimodal interaction grows. For example, in a multimodal image editor, a user can point to a person in a photo and say "*remove shadow on face*." However, the question remains: how does an end-user discover what they can say and how to say it?

In our work, we rely on two observations to enhance discoverability. First, by interactively selecting targets, non-speech modalities can help the end-user focus the high level question "what can I say?" to "what can I say *here* and *now*?" Second, our ability to contextually overlay text allows us to provide relevant command suggestions directly in the interface. Past work has shown that this type of feedback can enhance discoverability in NLIs [11, 19, 24]. Example commands can be made contextually relevant not only based on current workflows but also on explicit interaction. More broadly, we propose that the touch modality offers an opportunity for teaching speech commands. By providing examples relevant to the touch target, the interface performs double duty for discoverability.

*This work was done during the author's internship at Adobe Research.

The reality of multimodal interfaces, in particular when one modality (i.e., speech) relies on AI and can fail to perform as intended, is that end-users are likely to switch to a more reliable form (e.g., mouse or touch). While potentially less expressive, most GUIs also support directed exploration, which may be more comfortable or convenient than speech. In other words, while the modalities may be symbiotic, end-users may come to rely on one more than the other, further hindering discovery. Our approach, however, takes advantage of situations where non-speech interaction is used to provide speech examples. We can provide both immediately useful examples and create a 'teachable moment' for helping the user discover commands and phrasings useful in the future.

To better understand alternatives for command discovery, we implement a prototypical speech+touch image editor. In addition to more standard features (e.g., image filters), our prototype also supports entity recognition which creates referenceable objects. Thus, while supporting speech commands that reference the GUI (e.g., "*activate the draw tool*") our prototype also supports speech commands that reference image objects such as "*apply a sepia filter on the dog*" and multimodal commands with deictic phrasing: "*set the border size of this to 5.*"

To help users discover and learn this variety of speech commands, we present an approach for generating and ranking command suggestions. Using this approach, we implement three interface variants of the speech+touch image editor to present contextually-relevant command suggestions. The variants include an interface that presents suggestions using a standard list-based view, one that uses contextual overlay windows to present suggestions, and a third that embeds commands within the GUI. The proposed interfaces try to make users aware of *what can be done* and simultaneously teach them *how the commands can be invoked* (e.g., phrasing variants and multimodal interactions). To understand the pros and cons of the interfaces and assess if they aid discoverability, we conducted a preliminary user study online, allowing participants to use the system on their own devices without the presence of an experimenter. We report findings from the study highlighting observations and participant feedback indicating that in-situ command suggestions promote discovery and encourage the use of speech input.

2 RELATED WORK

Discoverability, as we define it, relates to two pieces of the broader learnability space [23]. Specifically, we are interested in awareness and understanding. These represent a long-standing challenge for speech-based interfaces [26, 37, 43, 46]. In fact, not knowing what commands are available or how to phrase them is the second most common obstacle faced by users of speech interfaces (after speech-recognition accuracy issues [31]). One common approach to address the lack of discoverability is to present users with lists of example commands as part of the onboarding experience or as notifications when new commands are available. For example, Apple's Siri [2] presents a list of example commands at start-up and lets users invoke the "Some things you can ask me" menu to see command examples on-demand. To make users aware of new or infrequently used commands, companies like Amazon and Xfinity send weekly emails to owners of their devices (Alexa and the voice-driven X1 set-top box, respectively) listing "New things you (users) can say." However, displaying example commands periodically or during onboarding may not be sufficient as users tend to forget these commands while performing tasks [14].

An alternative for supporting discovery is to directly present contextually-relevant commands while the end-user is interacting with the system. An 'always-on' approach might be an overlay panel displaying all commands that can be issued based on the active tool [24]. A more user-driven version is a "What Can I Say?" menu that shows commands relevant to the active application [11] on a mobile device (e.g., if the email application is open, the menu suggests commands like 'Compose Email', 'Reply'). Such an approach may be familiar to the users of the traditional 'F1' contextual help key [40]. For widgets represented only by graphical icons (i.e., ones without labels) numeric labels can be added dynamically to support reference [11]. More recently, Furqan et al. presented DiscoverCal [19], a speech-based calendar system that adaptively displays command templates depending on the current task (e.g., setting up a meeting) and previously issued commands. In our work, we extend these strategies but focus on both *generating* and *presenting* contextually relevant command suggestions in a multimodal interface. Our approach enables discovery by providing examples for *speech* when *non-speech* modalities are used (e.g., through touch) .

Our work also builds upon adaptive user interfaces [3, 16, 36] and contextual tool recommendations [29, 48]. Our focus is on extending this research but specifically to support discoverability. Adaptive interfaces employ varied strategies to tailor system functionality to a user [15]. Leveraging prior work, to suggest contextually relevant commands and help users incrementally explore commands, we employ both highlighting [41] and frequency-based adaptation [30] strategies. While most traditional adaptive interfaces focus on optimizing the 'local' experience of the end-user, examples for discoverability can take into account longer-term objectives (e.g., learning). That said, prior work on command recommendation has identified design criteria that are directly relevant to discoverability. For example, past research has found that command recommendations work better when displayed *in context* and are *self paced* [29]. We also let users discover commands at a their own pace by allowing them to incrementally explore commands pertaining to different parts of the interface.

3 DESIGN SPACE

For consistency in framing our design space, we define a few key terms using the motivating example of a multimodal image editor. Our application is built to support user utterances, or ***commands*** (e.g., "*Change fill color to red*") and linguistic variants, or ***phrasings*** (e.g., "*Make fill red*"). Commands are built around system actions, or ***operations*** (e.g., applying filters to an image, changing fill color, selecting tools). Most operations have modifiable ***parameters*** (e.g., filter names, color names, tool names) and operate on one or more ***targets*** (e.g., selected region of an image, a shape on the canvas). To support discoverability, our goal is to produce example natural language commands and phrasings.

The design space for producing and displaying examples requires answering a number of driving questions: when should we show examples? Which examples should we show? And where should we show them? As with most designs of this type, one decision is likely to impact another. For example, deciding *where* to show example commands may be constrained by the interface’s state *when* we show them. We note that while we suggest an extensive range of specific design approaches, there are many additional options.

A unique challenge for discoverability in intelligent user interfaces is that components of the system will invariably be unreliable. For example, speech recognition may fail entirely or make mistakes, and a user’s command phrasing may only be partially supported. Discoverability in these contexts requires improving the user’s awareness and understanding not only of the possible commands but also failure cases. Users may also expect and tolerate these failures. Paradoxically since failure happens frequently, it may be easier to model failure cases than in traditional interfaces.

3.1 When to show command suggestions?

During onboarding. Most if not all applications do onboarding of available tools and features when users first start the application. This is a natural time to expose users to the supported operations and commands. Unfortunately, context such as the user’s tasks or actions is missing. The generic onboarding experience often leads users to close these walkthroughs or forget the presented commands [14].

During a session. In order to guide users in performing their tasks and provide task-specific suggestions, prior work has explored suggesting commands while a session is in progress [11, 19, 24]. These suggestions can be explicitly requested by users (e.g., search) or the system can automatically suggest commands at preset intervals or based on specific user interactions (e.g., recommendations). Most modern applications shy away from proactive approaches and rely on reactive solutions, because interrupting users may be disruptive to the users’ workflow [17, 45].

As feedback for failed utterances. An important component in any multimodal system that supports speech input is providing feedback for input commands. Feedback messages offer another opportunity to teach natural language commands. For example, when a command fails, the feedback can include examples that correct the errors. However, interpreting user mistakes and system errors in a complex system is not easy, and offering effective feedback messages may require building a separate interpretation system altogether. Furthermore, if one could reliably guess the intent from an incorrect utterance it may simply be better to execute it.

3.2 What commands to suggest?

Number of suggestions. Selecting the “right” number of commands to suggest is influenced by many factors including interface design, screen size, user context, etc. A system could suggest an exhaustive list of commands to make users aware of what operations are possible and the different ways to phrase commands. However, a long list is hard to remember and may be overwhelming. Slight variants in phrasing or parameters may also be less useful, as the end-user can readily infer a broad pattern from a small number of examples. A ‘top N’ list of examples may be easier to interpret but may not expose users to all possibilities.

Relevance. Prior work has shown that commands suggested in the context of an application [11] or task [19, 46] aid discoverability. However, when there are multiple operations that are applicable, some operations may be more relevant than others. A simple approach is to suggest all contextually-relevant commands. Alternatively, suggestions can be weighted by frequency of use, required expertise, etc. A separate concern is the balance of “relevant now” versus “relevant later” [29]. Both have advantages for discoverability but may have different effectiveness (both based on task context and individual differences in learning and memory).

Complexity. With natural language input, users can specify multiple operations or parameter values in a single command. For instance, to insert a rectangle that is colored red, one might issue the commands “*Insert a rectangle*” and “*Change fill color to red*.” Alternatively, one can issue a single command to perform both operations (e.g.,“*Insert a red rectangle*”). While complex commands are more efficient, they can be longer and more difficult to learn and say. The effectiveness of complex or simple commands may be impacted by the end-user’s ability to isolate generalizable parts of the command and how much scaffolding they require. When to suggest simple commands, complex commands, or a combination likely varies by the target domain, command language, and users.

Variance in phrasings. Most NLIs strive to be as “natural” as possible by interpreting a broad range of linguistic variability. For example, “*color red*,” “*change fill to red*,” “*fill it with red*,” and “*make this red*” all may refer to the same command. In teaching people what a system can understand, it is important to consider how much variety to show. Showing a lot of variety may lead users to think the system is “smarter” than it actually is. Showing only one or two phrasings may lead to higher accuracy but may feel “unnatural.” The accuracy of the system in interpreting ambiguous or out-of-vocabulary language must naturally play a role in how phrasing variants are presented.

Variance in parameter values. When showing suggestions that include parameters, it is worth considering whether to vary parameter values within the suggestions. For example, the suggestion “*fill with red*” can vary with different color values. Varying parameter values shows users a range of applicable values but this variance may be confusing to those who expect consistent examples. An alternative is to show templates (e.g., “*fill with [color]*”), which highlights the parameters but does not offer specific values.

3.3 Where to display suggestions?

One common approach for display is through a fixed region of the interface (e.g., side panel). However, a side panel is easy to close and requires that users divert their attention from the task at hand [8]. As an alternative, contextual menus anchored by direct manipulation could help with visual attention but may occlude parts of the interface. A hybrid approach that uses both a panel and contextual menus may offer the best of both worlds. However, this may require more space and lead to unnecessary redundancy.

4 EXAMPLE APPLICATION: IMAGE EDITOR

We study discoverability of natural language commands in the context of a commonly used application, an image editor. We chose this domain because image editing is a popular task, has a variety of operations, and can be performed on mobile devices where speech interfaces are most applicable. Additionally, prior work has

Operation		Parameters	Applicable to	Phrasing Templates	Examples
Filter	Add Filter	*name, value	Canvas image, Entity, Region, Image tile	Apply a _____ filter on _____	Apply a grayscale filter on all cars
				Add a _____ _____ effect to the canvas	Add a heavy saturate effect to the canvas
				Add _____ _____ effect here	Add light morph effect here
				Apply _____ filter	Apply sepia filter
	Edit Filter	*name, *value	Canvas image, Entity, Region, Image tile	Set the value of the _____ filter on _____ to _____	Set the value of the sepia filter on the person to light
				Change _____ effect on _____ to _____	Change grayscale effect on all images to heavy
				Make _____ filter _____	Make morph filter light
				Set _____ effect to _____	Set saturate effect to heavy
	Remove Filter	*name	Canvas image, Entity, Region, Image tile	Remove the canvas _____ effect	Remove the canvas morph effect
				Delete the _____ filter on _____	Delete the sepia filter on the dog
				Delete the _____ effect on this	Delete the sepia effect on this
				Remove all filters	Remove all filters
Style	Fill Color	*color	Shape	Update the fill color on _____ to _____	Update the fill color on all rectangles to blue
				Change the color of _____ to _____	Change the color of the rectangle to red
				Fill _____	Fill orange
				Set fill to _____	Set fill to blue
	Border Color	*color	Shape	Change the border color of _____ to _____	Change the border color of the rectangle to green
				Set the stroke of _____ to _____	Set the stroke of all rectangles to black
				Make the stroke of this _____	Make the stroke of this red
				Color stroke _____	Color stroke green
	Border Size	*width	Shape	Update stroke size of _____ to _____	Update stroke size of all rectangles to eight
				Change the border thickness of _____ to _____	Change the border thickness of the rectangle to five
				Make stroke width _____	Make stroke width eight
				Set the border size of this to _____	Set the border size of this to five
Delete			Image tile, Region, Shape	Delete _____ from the canvas	Delete all drawn regions from the canvas
				Delete _____	Delete all images
				Delete	Delete
				Delete this	Delete this
Label		*newLabel	Entity, Image tile, Region, Shape	Set the label to _____	Set the label to sunny region
				Label this as _____	Label this as Ross
Copy		copyCount	Image tile, Shape	Create a copy of _____	Create a copy of the image
				Make _____ copies of _____ on the canvas	Make two copies of all rectangles on the canvas
				Create a copy	Create a copy
				Make _____ copies	Make six copies
Activate Tool		*toolName	Toolbar buttons	Highlight the entities on the canvas	Highlight the entities on the canvas
				Activate the draw tool	Activate the draw tool
Insert Shape		fillColor, strokeColor, strokeWidth	Canvas image	Add a rectangle on the canvas	Add a rectangle on the canvas
				Add a rectangle with fill _____ and stroke _____	Add a rectangle with fill red and stroke orange

Table 1: Supported operations in the prototype image editor along with parameters corresponding to each operation and target objects individual operations can be performed on. An asterisk (*) signifies a parameter is for mandatory for an operation. A sample set of phrasing templates and parameterized example commands are also shown.

extensively explored speech-based multimodal image editing interfaces, thus giving us a good initial set of features and interactions to consider [21, 25, 27, 35].

4.1 User interface

Figure 1 shows the user interface of our prototype. At the top (Figure 1A) is a read-only text box that displays the spoken command. Below that is a feedback region for showing system feedback for spoken commands. To the left of the canvas is a toolbar (Figure 1B) that allows users to select tools for drawing free-form regions (✎), rectangles (▢), import additional images as tiles (🖼), and to highlight labeled entities (e.g., person, car, dog–as specified by the end user or an object recognition system) (👁). To the right of the canvas is the properties panel (Figure 1D) that displays editing operations for selected objects. Users can perform operations such as adding or removing filters, changing fill and border colors, deleting objects etc. Table 1 shows a full list of supported operations.

4.2 Triggering speech input

We employ a "push-to-talk" technique to let users issue spoken commands and control when the system should start/stop listening. To trigger listening users can long press (hold > one second) on various locations in the interface: the input box, the talk button (🗣), the canvas background image and objects (e.g., shapes, entities), or the system panels (Figures 1B,D). The system listens as long as the finger is touching the screen. The microphone icon in the input box flashes red to indicate active listening (🎙).

4.3 Interpreting speech input

Similarly to prior systems, we use a combination of a template and lexicon-based parser to interpret speech [20, 27, 39]. Our prototype identifies the operations, targets and parameters of the spoken command by comparing the input to predefined templates. If the input does not match a template, the system tokenizes the command

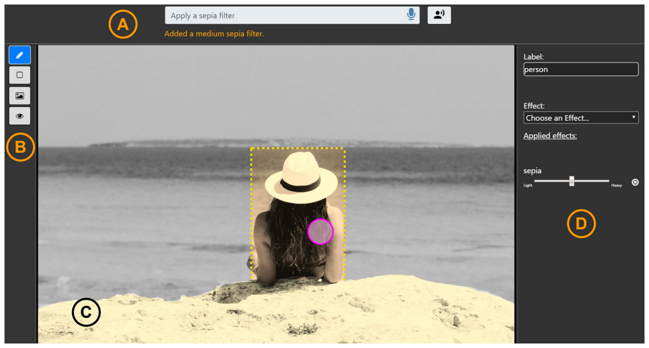


Figure 1: Prototype system's user interface. (A) Speech Input and Feedback Row, (B) Toolbar, (C) Canvas, and (D) Object Properties Panel. The pink circle shows the position of the user's finger. In this case, the user is pointing at the person in the canvas image and has issued the command "*Apply a sepia filter*". In response, the system infers that the user is referring to the lady in the image, creates a bounding box around her and adds a sepia filter.

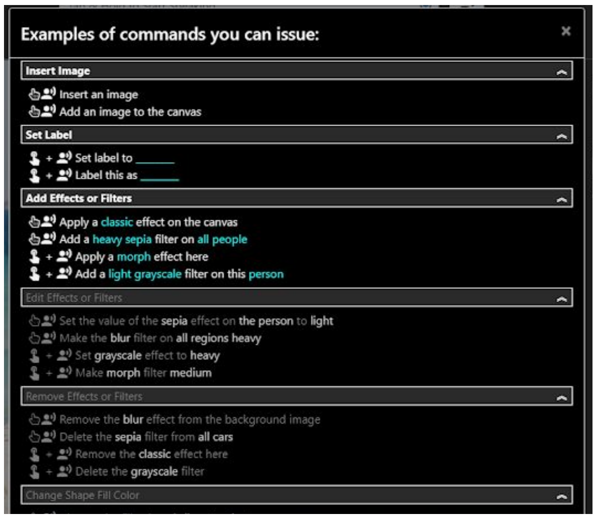
string and looks for specific keywords to infer the same information. In cases where the spoken command does not contain a target, the system infers the target through the interface state (e.g., which objects were previously selected) or touch input (e.g., what object was pointed at when the command was issued). Thus, touch is used to specify (or disambiguate) portions of the spoken command.

4.4 Implementation

We implemented the prototype as a web application primarily designed for tablet devices. To support entity labeling, we used OpenCV [6] and an open-source implementation of the MobileNet Single Shot Detector model trained on the COCO dataset [7, 28]. Through this, we detect entities and bounding boxes in the canvas image. We used the HTML5 WebKit speech recognition API and trained the recognizer with the system lexicon to improve detection of relevant keywords (e.g., filter and color names).

5 DISPLAYING CONTEXTUAL COMMAND SUGGESTIONS

After multiple design iterations and refinements, we selected three designs and implemented them as three separate interfaces—the *exhaustive interface*, the *adaptive interface*, and the *embedded interface*. All three help users discover commands in-situ but make different trade-offs and represent distinct points in the design space of command suggestions to aid command awareness and understanding. The exhaustive interface presents a list of all possible operations and example commands for each operation (Figure 2). The adaptive interface presents focused suggestions using contextual overlays that appear next to the finger when users touch the canvas or parts of the interface (Figure 3). Finally, the embedded interface presents suggestions next to *multiple* GUI elements (Figure 4). By varying when, where, and what example commands are displayed, the different interfaces explore ways to increase awareness of speech commands and how they relate to interface elements.



Examples of commands you can issue:

Insert Image
- Insert an image
- Add an image to the canvas

Set Label
- + Set label to ______
- + Label this as ______

Add Effects or Filters
- Apply a classic effect on the canvas
- Add a heavy sepia filter on all people
- + Apply a morph effect here
- + Add a light grayscale filter on this person

Edit Effects or Filters
- Set the value of the sepia effect on the person to light
- Make the blur filter on all regions heavy
- + Set grayscale effect to heavy
- + Make morph filter medium

Remove Effects or Filters
- Remove the blur effect from the background image
- Delete the sepia filter from all cars
- + Remove the classic effect here
- + Delete the grayscale filter

Change Shape Fill Color

Figure 2: A list of commands suggested in the Exhaustive Interface. In this case, commands for editing and removing filters are faded out because the user has not applied any filters yet.

5.1 Exhaustive interface

The exhaustive interface is modeled after traditional command menus that list example commands for all operations [2, 11, 19, 24] (see Figure 2).

When and where are suggestions presented?
Suggestions in this interface appear in a fixed position in the middle of the screen. Users can tap the talk button ( ) to see a list of all system operations and example commands for each operation. To aid readability, the examples are grouped by operation and users can collapse/expand operation groups.

What is suggested?
Although the interface displays multiple phrasings for all available operations, prior work exploring discoverability solutions for NLIs showed that contextualizing command suggestions aids discoverability [11, 19, 46]. Thus, the exhaustive interface fades out operations and commands that are not applicable to the active state of the interface. For example, if there are no shapes on the canvas, the commands for operations corresponding to shapes (e.g., *fill color*, *border size*) are faded out.

5.2 Adaptive interface

The exhaustive interface helps users discover a breadth of commands applicable to the active state of the interface. However, the modal window displaying the list occludes the entire interface and the long list of commands can be overwhelming. As an alternative, we implemented the adaptive interface based on prior designs using tooltips [12, 18, 22]. In our implementation we utilize tooltip-like overlays to suggest commands relating to the current touch target (see Figure 3). Because task-relevant suggestions are often more useful [11, 29], the adaptive interface tunes its suggestions to help

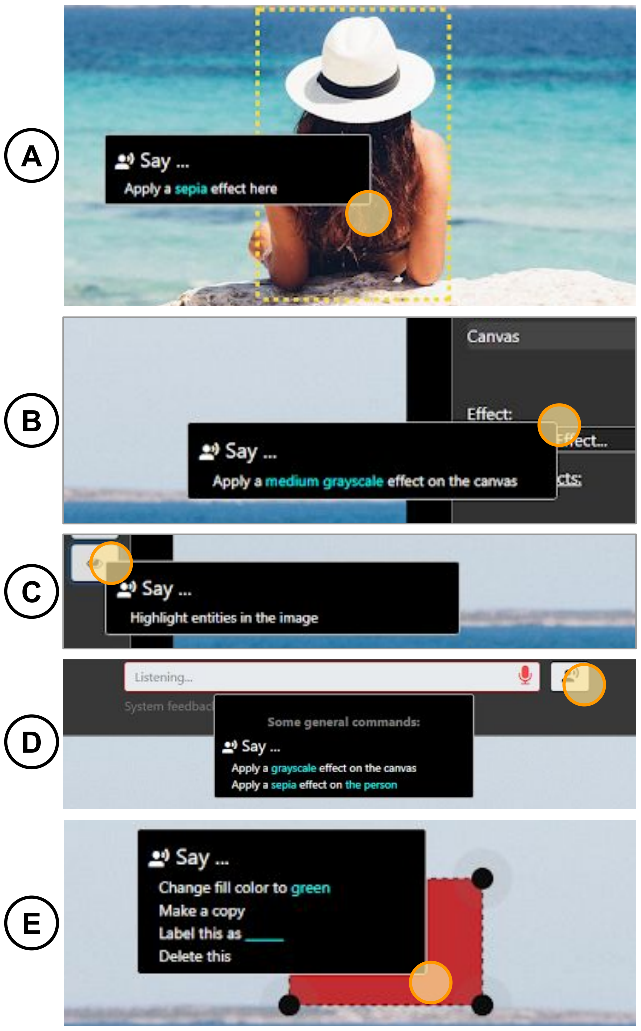

A
Say ...
Apply a sepia effect here
B
Canvas
Effect:
Say ...
Apply a medium grayscale effect on the canvas
C
Say ...
Highlight entities in the image
D
Listening...
System feedback
Some general commands:
Say ...
Apply a grayscale effect on the canvas
Apply a sepia effect on the person
E
Say ...
Change fill color to green
Make a copy
Label this as ____
Delete this

Figure 3: Adaptive interface. Suggested examples when the user touches (A) a person in the image, (B) the add filter drop-down in the properties panel, and (C) the entity detection button in the toolbar, (D) the talk button, and (E) a rectangle on the canvas.

users discover commands in their (predicted) workflow (i.e., their intended task).

When and where are suggestions presented?
To see command suggestions, users can long press on different parts of the interface including the canvas (Figure 3A,E), widgets and buttons in the properties panel and toolbar (Figure 3B,C), or the talk button (Figure 3D). Suggestions are presented through overlays next to the user's finger. Command suggestions may be specific to something directly under the user's finger (e.g., shape or image object) or may apply more generally to the interface. To avoid occlusion by the hand, the overlays appear above the user's finger on the canvas and are positioned to the left or right of the properties panel and the toolbar, respectively.

What is suggested?
Suggestions in this interface are contextual to the object under the user's finger. If the target is a widget, the suggestions are about the widget. If the user is touching the canvas, the suggestion will be about the object under the finger (e.g., background image, a person, shape, etc). For instance, suggestions for applying filters (e.g., "*Apply a grayscale filter*") may appear when users long press on the 'add effect' widget in the properties panel (Figure 3B) or on an object in the image (Figure 3A).

The system suggests one example command per applicable operation (Table 1). Command phrasings and parameter values vary over time. For example, the user might first see "*Apply a sepia effect here*" and later "*Add a morph filter*." To help users get accustomed to using speech, the system initially suggests simpler phrasings with fewer parameters and incrementally exposes users to more complex phrasings with multiple parameters as the user starts using the simpler phrasings. Figures 3A,B show this progression. Both suggestions map to the *add filter* operation but Figure 3B displays an example with both *filter name* and *filter strength* (grayscale, medium) whereas the example command in Figure 3A only includes the *filter name* (sepia). Long pressing the talk button at the top (🗣) shows suggestions of commands for the active canvas objects (e.g., shapes, background image, see Figure 3D).

In the adaptive interface, we experimented also with workflow-based suggestions. ***Workflow*** here is defined as the set of operations through which a user accomplishes specific, repeated, tasks. For example, if a user is transforming the image in Figure 3A to the image in Figure 1, the workflow will include operations for applying a grayscale filter on the image and applying a sepia filter on both the person and the rock the person is resting on. If the user is following a known workflow, the adaptive design restricts the number of suggestions it presents and prioritizes commands that align with the workflow. For instance, in Figure 3A, the system suggests a single command to apply the sepia filter because that is the next step in the predefined workflow. However, if no predefined workflow is available, the system goes with the default strategy of suggesting one command per applicable operation (Figure 3E).

Detecting whether a user is following a workflow automatically is a challenging and interesting problem but is not the focus of our work. Rather, we assume an underlying system that can make this prediction (e.g., [1, 13, 44]). For our experiments, we manually defined workflows for a number of source-target image pairs that allowed us to implement and evaluate the adaptive interface.

5.3 Embedded interface

Although the adaptive interface makes suggestions more contextual, it still uses overlays and thus occludes at least some part of the canvas with its tooltip-like overlays. To avoid occlusion entirely, the embedded interface 'augments' the GUI widgets with command suggestions instead of presenting them on the canvas next to the user's finger (see Figure 4). This approach has the added benefit of creating a visual mapping between GUI elements and speech commands. Because the suggestions appear throughout the GUI and incorporate the existing GUI labels, this interface presents phrasing templates rather than examples and does not use workflow information to select relevant suggestions.

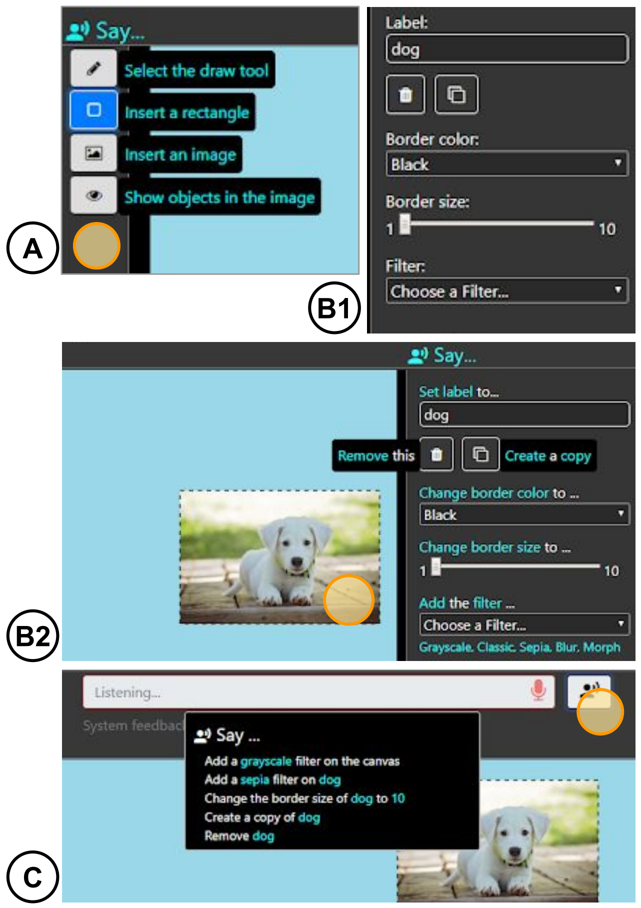


Figure 4: Embedded interface. (A) Commands suggested when the user holds the toolbar. (B1) Default state of the properties panel and (B2) Properties panel with embedded commands displayed when the dog image tile is pressed. (C) Commands suggested when the user holds the talk button.

When and where are suggestions presented?

To see suggestions, users can long press on different parts of the interface at any time. Command suggestions appear alongside the application GUI (Figure 4). For instance, if they long press on the canvas, the system displays commands within the properties panel (Figures 4B1,4B2). Blue text corresponding to speech commands augments the GUI widgets in the properties panel. To incrementally discover commands corresponding to the interface panels, users can also directly long press on the toolbar (Figure 4A) or the properties panel (rather than pressing only on canvas objects). Long pressing the talk button (🗣), displays *both* example commands corresponding to objects on the canvas (Figure 4C) and also embeds commands within the toolbar and properties panel.

What is suggested?

The embedded interface uses a combination of command templates and examples as part of its suggestions. Because this design augments the existing GUI widgets, which present values, it has to use *templates* instead of *examples* when suggesting commands in-situ. For instance, in Figure 4B2, the command template "*Change border color to ...*" appears next to the dropdown menu for changing the border color. To provide a consistent experience and give users confidence in how to talk, the system displays the same template throughout a session. Because the toolbar leaves little room to embed text commands, suggestions for the tools in the toolbar use command examples (not templates) similar to the adaptive interface. The examples presented when the user touches the talk button at the top (🗣) also follow the same approach as the adaptive interface.

6 GENERATING COMMAND SUGGESTIONS

We designed a single underlying approach to generate and rank contextually relevant commands in the three interfaces. The method provides an algorithmic scaffold for the described interfaces but can also be employed by other multimodal systems to suggest natural language commands.

Figure 5A provides an overview of our command suggestion approach. Given a target object, the system selects a subset of possible operations. Then, for each selected operation, the system chooses one phrasing template and populates the selected template with sample parameter values.

To select the subset of operations, the system considers object and workflow relevance. The system first filters out any operations that are not relevant to the target object (e.g., filters cannot be applied to shape objects). In the adaptive interface, the system then checks the predefined workflow to shortlist operations that are relevant to the user task. For instance, if a workflow specifies that the user needs to apply a filter on an image object, the system will suggest only *filter* commands even though other operations such as *copy* and *delete* may be applicable. Finally, the system ranks the filtered subset of operations using two factors: the number of times a speech command has been issued for an operation (*issued-count*) and the number of times an operation has previously been covered in the suggestions (*shown-count*). In our interfaces, we highly rank suggested operations with low issued-count and low shown-count to emphasize awareness of operations. That is, operations that are performed infrequently are suggested leading to discoverability not only of how to talk but also what is possible. An alternative might be to focus on high issued-count and high shown-count operations and display different phrasings. This approach emphasizes the learning of command variants for frequently used operations.

For each shortlisted operation, the system selects a phrasing from a set of phrasing templates (see examples in Table 1). The system ranks phrasing templates using input type, complexity, issued-count, and shown-count. Input type refers to whether the suggestions were invoked by touching on an object or the interface. When the touch target is a canvas object, phrasings with deictic references are ranked higher [33, 34]. Figure 5B shows the generation of a deictic phrasing template. The system picks the template "*Color this* ______," because the suggestion generation was invoked when the user pointed at a rectangle (the *input type*). *Complexity*, in our model refers to the number of parameters a template contains (e.g., complexity of *"Apply sepia effect"* is one whereas *"Apply heavy sepia effect"* is two.) The system ranks templates with lowest complexity highest at first. As users start using low complexity phrasings, the system increases the complexity of the highest ranked template.

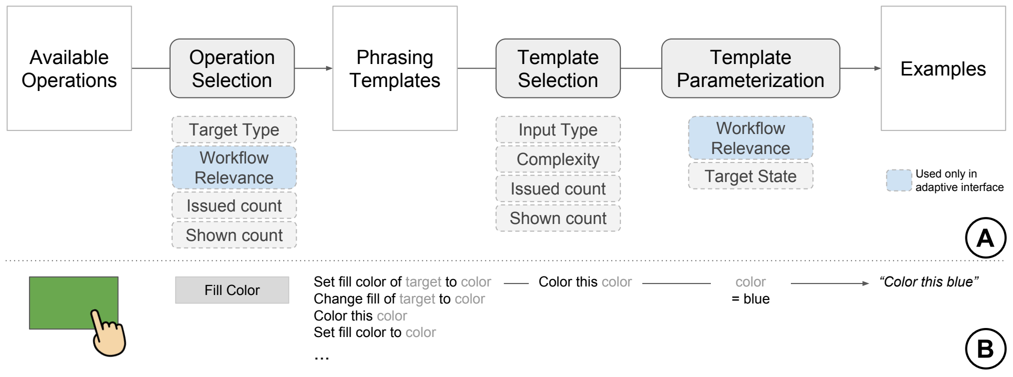


Figure 5: (A) Overview of the command suggestion approach and (B) Example execution for *Fill Color* when the user points to a shape.

Whenever an operation is performed twice, the system increases the complexity of the selected phrasing by one. Thus, users are incrementally exposed to complex spoken commands as they learn to perform operations. The issued-count and shown-count factors are as described above, and specifically refer to use through speech and presentation as a suggestion. In the implementation for all three interfaces the system ranks phrasings with low issued-count (use) and low shown-count (presentations) higher in order to expose users to a wider range of phrasings.

As a final step, the system parameterizes the selected phrasing template to generate an example command. When the system is aware of the user's task, it will select parameters that are workflow-oriented. Otherwise, it selects parameter values that are *different* from the target's current values. For instance, in Figure 5B, the suggested fill command when touching the green rectangle would include colors other than green.

6.1 Failure inference

Early pilot studies with the system showed us that command suggestions alone were not sufficient, because users looked to error messages to understand why their command did not work. This led us to develop a feedback mechanism that incorporates suggestions. In all three interfaces, the feedback region below the text box (Figure 1A) also suggests example commands as part of feedback to the user. To suggest examples in this context, the system infers the cause of command failure using heuristics and classifies each failure as a phrasing error, a parameter error, an operation-object mapping error, or an operation recognition error. *Phrasing errors* are identified as commands that contain a valid parameter but are inconsistent with the grammar or lack keywords (e.g., "*Make sepia*"). In such cases, the system suggests an example command using that parameter value (e.g., "*Add a sepia filter*"). A *parameter error* is determined if there is a valid operation but a missing or unsupported parameter value (e.g., "*Change fill color*" or "*Add the retro filter*"). For this, the feedback indicates that the command is incomplete and displays a list of supported values with an example (e.g., "*Change fill color to green*"). A third error type, *operation-object mapping error*, is when the system infers both operation and parameters but the command is targeted on an unsupported object (e.g., saying "*Apply a morph filter*" while pointing on a rectangle). In this case, the feedback lists the applicable object types (i.e., images in this example above). Finally, if the system is neither able to infer the operation nor the parameter in a command (e.g., "*undo*"–which was not implemented in our prototype) the system counts this as an *operation recognition error* and indicates to the user that they should try one of the offered examples.

6.2 Generalizability of the approach

We have presented the command suggestion approach (Figure 5A) in the context of our prototype system. However, the approach itself is not specific to image editing tools and can be used by other multimodal systems to enhance speech discoverability. To implement suggestions we primarily rely on mappings between operations and objects (Table 1) and ranking factors (listed in Figure 5A). Most multimodal systems inherently define mappings between operations and targets. A text editor, for example, might allow operations such as changing the font size or color on sentences, paragraphs, etc. (the target objects). The factors used to select operations and phrasings are also generalizable. For example, we might change ranking to *reinforce* learning by suggesting commands that have been issued before rather than novel examples. The only required change would be modifying the ranking to prefer high issued-counts. Likewise, to suggest complex examples instead of simpler ones, ranking weights can be modified to prefer higher complexity (e.g., commands with more parameters) when selecting phrasing templates.

7 PRELIMINARY USER STUDY

We conducted a preliminary user study to assess if the command suggestions helped users discover and use speech commands. The study also helped us understand if there were preferences between the three interfaces. We followed a between-subjects design and ran the study on UserTesting [42], an online platform commonly used for testing web and mobile applications. Deploying an online study allowed us to evaluate how the designs would work 'in the wild' where people use their own computers and receive minimal training. UserTesting participants are walked through a step-by-step guided experience with questions and prompts to use a website or interface. Throughout the process participants follow a think-aloud study methodology, and both audio and screen captured video are recorded through the UserTesting platform. All sessions are shared as video recordings.

7.1 Participants and setup

After piloting, we posted three individual studies requesting eight participants for each condition (Exhaustive: X1-X8, Adaptive: A1-A8, Embedded: E1-E8). The participants were paid $10 for each session. To qualify, participants had to 1) be native English speakers, 2) use a Microsoft Surface Pro tablet without an external keyboard, mouse, or stylus, and 3) run the study on the Google Chrome browser. We did not require any prior experience with image editing tools or speech systems.

Of the 24 sessions, we discarded 7 (2 exhaustive, 3 adaptive, 2 embedded) where we witnessed hardware problems (e.g., speech or touch not working) or when the participants did not follow the study instructions (e.g., used a mouse, did not attempt all tasks). We report our results based on the 17 completed sessions (6 exhaustive, 5 adaptive, 6 embedded). The 17 participants (4 males, 13 females) were 19-48 years old. Seven participants self-identified as novices with photo-editing tools (Exhaustive: 1, Adaptive: 4, Embedded: 2), eight as intermediate (Exhaustive: 4, Embedded: 4), and two rated themselves as experts (Exhaustive: 1, Adaptive: 1). Of the 17, three participants self-identified as novices with speech interfaces (Adaptive: 2, Embedded: 1), four self-identified as intermediate (Exhaustive: 3, Embedded: 1), and ten indicated high comfort with speech interfaces and regular use (Exhaustive: 3, Adaptive: 3, Embedded: 4).

7.2 Procedure

Study sessions lasted 22-48 minutes (32 minute mean and median). Each began with a questionnaire asking participants about their level of experience with photo editing tools (e.g., Photoshop) and speech interfaces (e.g., Siri, Alexa). Participants then watched two training videos (~ 4 min). The first video was common across the three designs and gave a general overview of the system, showing participants the different interface components and tools. The second video was design specific and showed participants how to talk to the application and invoke the command suggestions. Participants watched the videos in sequence and could not proceed until they watched both videos completely. The participants were then asked to complete three 'before-after' image editing tasks that required editing a source image (the 'before') to look as similar as possible to the target (the 'after'). The participants were asked to think aloud as they worked and did not receive time constraints. After completing the three tasks, participants answered a post-session questionnaire asking about their experience with the tool, their perceptions of the suggestions, and any additional feedback.

We analyzed the video recordings for each session, taking notes on participant behavior and manually curated all spoken commands for further success/failure analysis.

7.3 Results

The study resulted in a total of 834 spoken commands with an average of 49 spoken commands per participant across conditions. We did not have enough data to assess statistical significance, but we found that participants in the exhaustive condition used fewer commands (mean: 35, median: 37, median absolute deviation, or MAD: 16) than in the adaptive (mean: 52, median: 64, MAD: 19) and embedded (mean: 61, median: 50, MAD: 21) conditions indicating that the proposed overlay and augmentation-based suggestions may encourage participants to talk more.

Out of the 834 commands, 44% (369/834) failed. Among these, 39% (144/369) were speech-to-text transcription errors, 16% (59/369) were commands that got recorded while participants were thinking aloud, 10% (36/369) were cases where participants did not record the entire command before lifting their finger from the screen, 18% (65/369) were phrasing errors, 7% (28/369) were operation-object mapping errors, 5% (19/369) were operation recognition errors, and the remaining 5% (18/369) were parameter errors. In total, 35% (130/369) of the errors were connected to discoverability of *what* to say and *how* to say it. The high number of speech-to-text errors show that transcription remains one of the major obstacles for speech interfaces [31]. However, the relatively lower number of phrasing, parameter, operation-mapping, and operation recognition errors suggest that the interfaces helped participants learn what they could say. In fact, despite the high number of speech-to-text errors, 13/17 participants (exhaustive: 4/6, adaptive:5/5, embedded: 4/6) said that the suggestions helped them learn how to talk to the system and 10/17 participants (exhaustive: 3/6, adaptive:4/5, embedded: 3/6) said that the suggestions encouraged them to talk. For instance, A5 said, "*when I push on something it said say this and so instead of like trying to scroll through or do something else I just said what it told me to say.*" Explaining the importance of suggestions in addition to onboarding videos, X2 said "*it's absolutely critical to have that cheat sheet there. Maybe I didn't pay as much attention as I should have to the videos ... but there's your average user. You're not going to have their complete attention anyway even if they watch the tutorial videos*"

The command success rate across designs was comparable with average command success rate per session in the high 70s% (exhaustive=77%, adaptive=77%, embedded=79%). This is particularly encouraging for the adaptive and embedded interface where the participants tended to speak more. For this analysis, we exclude speech-to-text errors, conflicts with the think-aloud protocol, and errors from lifting the finger before completing a command, because they affected all conditions.

Varying distribution of deictic speech commands

Deictic commands (e.g., "*make this green*", "*add a morph filter here*") are common in multimodal systems. However, users can also point at objects and say a command without including a deictic word (e.g., saying "*Make fill blue*" while pointing on a shape). Since these are not strictly deictic commands, we refer to both commands that include deictic references and non-deictic commands issued while pointing at a specific object as deictic++ commands. We expected to see most deictic++ commands in the adaptive interface because suggestions appear next to the user's finger. The logs confirm this to be the case. In the adaptive interface, deictic++ commands accounted for 74% of successful commands. In the exhaustive interface deictic++ commands accounted for 69% of successful commands. And in the embedded interface they composed 43% of successful commands. The low number of deictic++ commands in the embedded interface was a surprise. We think this low number may be due to three factors. First, users of the embedded interface saw very few deictic examples. Since this interface was designed for fewer examples with consistent phrasing and the commands had to be embedded next to the GUI widgets constraining the types of examples that could be shown. Second, participants may have missed some of the suggestions in the panel because they were focused on editing the image and the suggestions were on the side. Finally, the dense nature of embedded suggestions may make them difficult to read. E2 was looking for a command to change the border size of a rectangle and invoked the panel suggestions to see potential commands. However, since the command was surrounded by other UI widgets and commands, she did not see the suggestion corresponding to the border size operation and invoked the suggestions multiple times.

Using failure feedback for discovery

All participants used the examples shown in the feedback interface at least once. Some participants used the feedback itself as a discovery mechanism. For instance, one participant (X5) tried using the feedback as a command search mechanism. After noticing that the system gave examples in the feedback interface, she started issuing single word commands such as "*copy*" and "*border*" to see the commands suggested by the system. The feedback example commands seemed most effective when they were linguistically similar to the failed input command or used the same parameter values. For example, when X2 uttered an ill-phrased command to color a rectangle blue, the feedback suggested the command "*Set the fill color to blue*." Reading this the participant said "*Thank you!*" and reuttered the command, almost as if it was a dialog between him and the system. The frequent use of system feedback coupled with such behavior highlights the value in exploring feedback techniques for command discovery. Potential avenues include presenting confirmatory questions or explanatory messages to assist error recovery and command discoverability [4, 38].

Suggestions do not overcome lack of domain knowledge

Even participants who used many speech commands would revert to using only touch whenever they were unsure of terminology. For example, applying filters requires knowing the names of the filters. Some participants did not know which filter names corresponded to which types of visual changes. So while exploring, they used the drop down menu and tried out different filters using touch rather than giving speech commands. More experienced participants switched back to using speech after learning the available filters while novice participants continued to use the drop down menu. The lack of a visual preview hindered exploration and learning, in particular around tools that have names that are hard to interpret. How to support this kind of exploration with speech interfaces remains an open question.

Challenges in generalizing example commands

While most participants realized that the suggested commands were only examples and they could modify parameter values when issuing commands, some were initially confused and thought the examples were recommendations of things they *must* say. For example, E4 read out all of the system suggestions leading to her uttering a total of 147 commands compared to an average of 43 commands per session across the remaining 16 participants. Describing her actions, she said "*honestly I'm just doing whatever this directs me to do. It's really difficult for me to know what to say here.*" indicating that she interpreted the command examples as required next steps. While the directed nature of the study protocol and UserTesting platform may have exacerbated this confusion, a potential risk for future systems to consider is that users may confuse contextual examples (i.e., what is possible at a given instant) with system recommendations (i.e., what to do next).

No single design fits all

To become more aware of available commands and learn supported phrasings, different people need different types of suggestions. For instance, the participant in the embedded condition (E4) who kept parroting the suggested commands without realizing they were only examples would have benefited from the workflow-based suggestions of the adaptive interface. On the other hand, one of the participants in the adaptive condition (A5) often had to invoke overlay windows for specific widgets in the properties panel because the suggestions she saw on the canvas did not always show commands for the styling operations she wanted to perform. For such participants, it may be useful to have suggestions both as a contextual overlay next to the finger and embedded in the properties panel. Hence, while we separated the three designs for the purpose of the study, in the future it may be worthwhile to explore how strategies from the different designs can be combined.

8 LIMITATIONS AND FUTURE WORK

The preliminary user study helped us collect valuable feedback. An important next step is completing a formal statistical evaluation comparing the different designs using a larger pool of participants. A more formal study would help validate our current findings and gain a deeper understanding of the effects of the interface features on task performance and learning. Additionally, applying this approach to a more complex application with more tools and operations will invariably help us understand how to balance breadth of examples with GUI-widget-to-speech-command mappings.

An inherent limitation with the approach of suggesting commands is that the suggestions may give users a false sense of the system's interpretation capabilities. In other words, users may think that the system can only understand commands if they are phrased exactly as they are in the suggestions. While users following suggested phrasings may have a positive impact on system performance, it can also limit what users might try to accomplish with a system. Hence, another potential area for improvement is to explore more automated techniques to generate a broader set of phrasing templates that cover a wider range of system capabilities.

In our current implementation, the command suggestions cover variation in phrasings, parameters, and complexity. However, there are classes of commands that we do not support in our current work including follow-up commands and speech+gesture commands. Similar to deictic commands, follow-up commands have different linguistic phrasings (e.g., "*Set fill to blue*" > "*Now green*"). Supporting follow-up commands requires keeping track of previously executed operations. And while it may be straightforward to add support for discoverability of simple follow-up commands like "*now green*," it is less clear how to support follow-up commands that combine multiple previous operations (e.g., "*do this again 4 times*") where *this* may refer to multiple operations like changing a border, fill, and resizing. Another class of commands we have yet to explore are speech+gesture commands. For example, as a user drags the corner of rectangle, a system can support snapping with speech commands. Or, as shown by Laput et al. [27], users can say "*blur in this direction*" while performing a gesture to indicate the direction of the blur. An open area for future work is helping users discover these classes of commands through examples.

9 CONCLUSION

In this work we explored alternative designs for enhancing discoverability of speech commands in multimodal interfaces. We described a framework to generate and rank command examples and used it to implement three variants of a prototypical touch+speech image editor: *exhaustive, adaptive, and embedded.* Based on an online user study emulating 'in the wild' system usage, we described how contextual command suggestions promoted discovery in all interfaces and encouraged the use of speech input. We found that users uttered more deictic commands in the adaptive interface and that system feedback could be used to enhance discovery and mitigate challenges participants had in generalizing from examples. Our framework for deciding how, when and which suggestions to show generalizes beyond photo editing tools and can be adapted to enhance discoverability of natural language commands in a variety of multimodal interfaces.

10 ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers and our study participants for their time and helpful feedback. We also thank Celso Gomes for his feedback on the prototype and help in setting up the study.

REFERENCES

[1] Eytan Adar, Mira Dontcheva, and Gierad Laput. 2014. CommandSpace: Modeling the Relationships Between Tasks, Descriptions and Features. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14).* ACM, New York, NY, USA, 167–176. https://doi.org/10.1145/2642918.2647395

[2] Apple Siri 2018. https://www.apple.com/siri/.

[3] David Benyon. 1993. Adaptive systems: a solution to usability problems. *User modeling and User-adapted Interaction* 3, 1 (1993), 65–87.

[4] Dan Bohus and Alexander I. Rudnicky. 2008. *Sorry, I Didn't Catch That!* Springer Netherlands, Dordrecht, 123–154. https://doi.org/10.1007/978-1-4020-6821-8_6

[5] Richard A Bolt. 1980. *"Put-that-there": Voice and gesture at the graphics interface.* Vol. 14. ACM.

[6] Gary Bradski and Adrian Kaehler. 2000. OpenCV. *Dr. Dobb's journal of software tools* 3 (2000).

[7] Caffe implementation of Google MobileNet SSD detection network 2018. https://github.com/chuanqi305/MobileNet-SSD.

[8] Paul Chandler and John Sweller. 1992. The split-attention effect as a factor in the design of instruction. *British Journal of Educational Psychology* 62, 2 (1992), 233–246.

[9] Pei-Yu (Peggy) Chi, Daniel Vogel, Mira Dontcheva, Wilmot Li, and Björn Hartmann. 2016. Authoring Illustrations of Human Movements by Iterative Physical Demonstration. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16).* ACM, New York, NY, USA, 809–820. https://doi.org/10.1145/2984511.2984559

[10] P. R. Cohen, M. Dalrymple, D. B. Moran, F. C. Pereira, and J. W. Sullivan. 1989. Synergistic Use of Direct Manipulation and Natural Language. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '89).* ACM, New York, NY, USA, 227–233. https://doi.org/10.1145/67449.67494

[11] Eric Corbett and Astrid Weber. 2016. What Can I Say?: Addressing User Experience Challenges of a Mobile Voice User Interface for Accessibility. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '16).* ACM, New York, NY, USA, 72–82. https://doi.org/10.1145/2935334.2935386

[12] Yibo Dai, George Karalis, Saba Kawas, and Chris Olsen. 2015. Tipper: contextual tooltips that provide seniors with clear, reliable help for web tasks. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems.* ACM, 1773–1778.

[13] Himel Dev and Zhicheng Liu. 2017. Identifying Frequent User Tasks from Application Logs. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces (IUI '17).* ACM, New York, NY, USA, 263–273. https://doi.org/10.1145/3025171.3025184

[14] Jinjuan Feng, Clare-Marie Karat, and Andrew Sears. 2004. How productivity improves in hands-free continuous dictation tasks: lessons learned from a longitudinal study. *Interacting with computers* 17, 3 (2004), 265–289.

[15] Leah Findlater and Krzysztof Z Gajos. 2009. Design space and evaluation challenges of adaptive graphical user interfaces. *AI Magazine* 30, 4 (2009), 68.

[16] Leah Findlater and Joanna McGrenere. 2004. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI conference on Human factors in computing systems.* ACM, 89–96.

[17] James Fogarty, Scott E Hudson, Christopher G Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny C Lee, and Jie Yang. 2005. Predicting human interruptibility with sensors. *ACM Transactions on Computer-Human Interaction (TOCHI)* 12, 1 (2005), 119–146.

[18] Adam Fourney, Ben Lafreniere, Parmit Chilana, and Michael Terry. 2014. InterTwine: creating interapplication information scent to support coordinated use of software. In *Proceedings of the 27th annual ACM symposium on User interface software and technology.* ACM, 429–438.

[19] Anushay Furqan, Chelsea Myers, and Jichen Zhu. 2017. Learnability Through Adaptive Discovery Tools in Voice User Interfaces. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '17).* ACM, New York, NY, USA, 1617–1623. https://doi.org/10.1145/3027063.3053166

[20] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. 2015. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software &#38; Technology (UIST '15).* ACM, New York, NY, USA, 489–500. https://doi.org/10.1145/2807442.2807478

[21] Arno PJ Gourdol, Laurence Nigay, Daniel Salber, Joëlle Coutaz, et al. 1992. Two Case Studies of Software Architecture for Multimodal Interactive Systems: VoicePaint and a Voice-enabled Graphical Notebook. *Engineering for Human-Computer Interaction* 92 (1992), 271–84.

[22] Tovi Grossman and George Fitzmaurice. 2010. ToolClips: an investigation of contextual video assistance for functionality understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 1515–1524.

[23] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. 2009. A Survey of Software Learnability: Metrics, Methodologies and Guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09).* ACM, New York, NY, USA, 649–658. https://doi.org/10.1145/1518701.1518803

[24] Susumu Harada, Jacob O Wobbrock, and James A Landay. 2007. Voicedraw: a hands-free voice-driven drawing application for people with motor impairments. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility.* ACM, 27–34.

[25] Alexander G Hauptmann. 1989. Speech and gestures for graphic image manipulation. *ACM SIGCHI Bulletin* 20, SI (1989), 241–245.

[26] Laurent Karsenty. 2002. Shifting the design philosophy of spoken natural language dialogue: From invisible to transparent systems. *International Journal of Speech Technology* 5, 2 (2002), 147–157.

[27] Gierad P Laput, Mira Dontcheva, Gregg Wilensky, Walter Chang, Aseem Agarwala, Jason Linder, and Eytan Adar. 2013. Pixeltone: A multimodal interface for image editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 2185–2194.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision.* Springer, 740–755.

[29] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. 2009. CommunityCommands: command recommendations for software applications. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology.* ACM, 193–202.

[30] Jeffrey Mitchell and Ben Shneiderman. 1989. Dynamic versus static menus: an exploratory comparison. *ACM SigCHI Bulletin* 20, 4 (1989), 33–37.

[31] Chelsea Myers, Anushay Furqan, Jessica Nebolsky, Karina Caro, and Jichen Zhu. 2018. Patterns for How Users Overcome Obstacles in Voice User Interfaces. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* ACM, 6.

[32] Sharon Oviatt. 1997. Multimodal interactive maps: Designing for human performance. *Human-computer interaction* 12, 1 (1997), 93–129.

[33] Sharon Oviatt. 1999. Ten myths of multimodal interaction. *Commun. ACM* 42, 11 (1999), 74–81.

[34] Sharon Oviatt, Antonella DeAngeli, and Karen Kuhn. 1997. Integration and synchronization of input modes during multimodal human-computer interaction. In *Referring Phenomena in a Multimedia Context and their Computational Treatment*. Association for Computational Linguistics, 1–13.

[35] Randy Pausch and James H. Leatherby. 1991. An Empirical Study: Adding Voice Input to a Graphical Editor. *Journal of the American Voice Input/Output Society* 9 (1991), 2–55.

[36] Tim F. Paymans, Jasper Lindenberg, and Mark Neerincx. 2004. Usability Trade-offs for Adaptive User Interfaces: Ease of Use and Learnability. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04)*. ACM, New York, NY, USA, 301–303. https://doi.org/10.1145/964442.964512

[37] Ben Shneiderman and Pattie Maes. 1997. Direct manipulation vs. interface agents. *interactions* 4, 6 (1997), 42–61.

[38] Gabriel Skantze. 2005. Exploring human error recovery strategies: Implications for spoken dialogue systems. *Speech Communication* 45, 3 (2005), 325–341.

[39] Arjun Srinivasan and John Stasko. 2018. Orko: Facilitating multimodal interaction for visual exploration and analysis of networks. *IEEE transactions on visualization and computer graphics* 24, 1 (2018), 511–521.

[40] Keith Taylor. 1990. IBM systems application architecture: common user access from first principles. *Computing & Control Engineering Journal* 1, 3 (1990), 123–127.

[41] Theophanis Tsandilas et al. 2005. An empirical assessment of adaptation techniques. In *CHI'05 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2009–2012.

[42] UserTesting.com, Inc. 2018. https://www.usertesting.com/.

[43] Marilyn A. Walker, Jeanne Fromer, Giuseppe Di Fabbrizio, Craig Mestel, and Don Hindle. 1998. What Can I Say?: Evaluating a Spoken Language Interface to Email. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '98)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 582–589. https://doi.org/10.1145/274644.274722

[44] Xu Wang, Benjamin Lafreniere, and Tovi Grossman. 2018. Leveraging Community-Generated Videos and Command Logs to Classify and Recommend Software Workflows. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 285, 13 pages. https://doi.org/10.1145/3173574.3173859

[45] Jun Xiao, Richard Catrambone, and John Stasko. 2003. Be quiet? evaluating proactive and reactive user interface assistants. In *Proceedings of INTERACT*, Vol. 3. 383–390.

[46] Nicole Yankelovich. 1996. How do users know what to say? *interactions* 3, 6 (1996), 32–43.

[47] 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 (W4A '14)*. ACM, New York, NY, USA, Article 36, 4 pages. https://doi.org/10.1145/2596695.2596720

[48] S. Zolaktaf and G. C. Murphy. 2015. What to Learn Next: Recommending Commands in a Feature-Rich Environment. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. 1038–1044. https://doi.org/10.1109/ICMLA.2015.55