ScratchThat: Supporting Command-Agnostic Speech Repair in Voice-Driven Assistants

JASON WU, Carnegie Mellon University, USA
KARAN AHUJA, Carnegie Mellon University, USA
RICHARD LI, Georgia Institute of Technology, USA
VICTOR CHEN, Georgia Institute of Technology, USA
JEFFREY BIGHAM, Carnegie Mellon University, USA

Speech interfaces have become an increasingly popular input method for smartphone-based virtual assistants, smart speakers, and Internet of Things (IoT) devices. While they facilitate rapid and natural interaction in the form of voice commands, current speech interfaces lack natural methods for command correction. We present ScratchThat, a method for supporting command-agnostic speech repair in voice-driven assistants, suitable for enabling corrective functionality within third-party commands. Unlike existing speech repair methods, ScratchThat is able to automatically infer query parameters and intelligently select entities in a correction clause for editing. We conducted three evaluations to (1) elicit natural forms of speech repair in voice commands, (2) compare the interaction speed and NASA TLX score of the system to existing voice-based correction methods, and (3) assess the accuracy of the ScratchThat algorithm. Our results show that (1) speech repair for voice commands differ from previous models for conversational speech repair, (2) methods for command correction based on speech repair are significantly faster than other voice-based methods, and (3) the ScratchThat algorithm facilitates accurate command repair as rated by humans (77% accuracy) and machines (0.94 BLEU score). Finally, we present several ScratchThat use cases, which collectively demonstrate its utility across many applications.

CCS Concepts: • **Human-centered computing → Sound-based input / output**; *Interaction techniques*; *Ubiquitous and mobile devices.*

Additional Key Words and Phrases: Conversational Agents, Speech Interfaces, Voice User Interfaces, Error Correction, Dialog Interaction, Speech Repair

ACM Reference Format:
Jason Wu, Karan Ahuja, Richard Li, Victor Chen, and Jeffrey Bigham. 2019. ScratchThat: Supporting Command-Agnostic Speech Repair in Voice-Driven Assistants. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 2, Article 63 (June 2019), 17 pages. https://doi.org/10.1145/3328934

1 INTRODUCTION

Speech interfaces, particularly ones used in voice-driven virtual assistants have become increasingly popular as the accuracy of automatic speech recognition has improved [8]. Despite natural language being a rich and intuitive form of interaction, modern smart assistants fail to deliver many of the affordances expected of natural conversation [17, 32, 36]. One straightforward example of this shortcoming is speech interfaces do not allow

Authors' addresses: Jason Wu, Carnegie Mellon University, USA, jsonwu@cmu.edu; Karan Ahuja, Carnegie Mellon University, USA, kahuja@cs.cmu.edu; Richard Li, Georgia Institute of Technology, USA, lichard49@gatech.edu; Victor Chen, Georgia Institute of Technology, USA, vchen36@gatech.edu; Jeffrey Bigham, Carnegie Mellon University, USA, jbigham@cs.cmu.edu.

2474-9567/2019/6-ART63 $15.00
https://doi.org/10.1145/3328934

for natural speech repairs, *i.e.*, changing some detail of a query after making it. While users can change queries in text-based conversational interfaces by editing the text they entered, such functionality is unavailable via time-dependent speech. Instead, users generally must cancel a prior command explicitly and then speak the whole statement again, which is time-consuming and cumbersome. In this paper, we present ScratchThat, a command-agnostic speech repair system for voice driven virtual assistants.

ScratchThat uses an algorithm that we introduce to find the most probable replacement for command parameters, and then makes the replacement. For concreteness, consider the following user query:

> "OK Google, order a pizza from *Pizza Hut*....actually make that *Papa John's*."

Voice assistants, such as Google Assistant, will misinterpret the sentence or not take into account the correction clause without explicit error handling programmed into each skill [1]. However, ScratchThat is able to pick up the change in sentence structure and correctly recognize the query as ordering a pizza from Papa John's. Overall, the paper makes the following contributions in the area of voice-driven interfaces:

(1) We have developed a command-agnostic speech repair system. Current mechanisms for correction are expected to be explicitly programmed into the app to handle specific cases [1]. In contrast, our approach can be added as a layer on top of command logic and is not dependent on the expected query structure of a certain command. We do not require any training data or examples of commands for training replacements.
(2) We showcase a more generalizable system. Current speech repair models make assumptions about the placement of edit terms (such as "actually", "I mean" or "instead") [21–24, 37, 45], which we find to be limiting. Our system is free from such assumptions and showcases performance on a diverse range of commands. Moreover, unlike many previous approaches, our system can handle multiple out-of-order repair clauses in a sentence.
(3) We ran a comprehensive suite of experiments that evaluated speed and usability across 4 repair methods, encompassing 120 crowd workers and 10 live participants. Our results provide insights into speech repair systems that generalize beyond our implementation.

2 RELATED WORK

2.1 Conversational Agents

While early research on conversational agents was mainly centered around sustaining believable conversation [29, 41], the recent research focus has shifted towards creating functional interfaces [20, 33, 38] and conversational modeling [18, 35, 40].

Advances in deep neural modeling techniques have allowed some relatively simple text-based conversational agents (*e.g.,* question answering agents and technology helpdesk agents) to be generated with end-to-end training [35, 40]. These models can be further improved by providing contextual information such as user web history [19], visual input [14], and affective cues [18, 43].

More complex conversational agents often define a framework for extending the agent's functionality with new commands [15]. Many consumer devices such as the Google Assistant, Amazon Alexa, and Apple's Siri include APIs for creating "skills", which allow users to perform various supported actions, control connected devices, and query information from the internet.

For such systems that potentially host thousands of "skills" with varying abilities and syntax, it is important for the natural language understanding component to ensure their expected functionalities. In our paper, we take this into consideration for ScratchThat.

2.2 Usability of Voice Interfaces

Speech recognition has recently reached the stage of enabling fast interactions with computers [34]. On the other hand, while smart voice assistants such as Google Assistant and Amazon Alexa have become popular in the consumer market [8], the naturalness of interacting with them is often still restricted [17, 32].

Many of the challenges related to voice interfaces stem from their invisibility which can cause difficulty for command discoverability and learnability [17, 30]. Often, voice interfaces support a large number of actions, but it is infeasible or impossible to provide users with hints on the currently available commands and their syntax.

Recently, researchers have sought to address this by applying principles of adaptive interface design to voice interfaces [17, 20, 26, 31], which allow interfaces to dynamically adjust for real-time behavior of users, often taking into account user familiarity [31]. In conjunction with user data-driven systems that provide augmented understanding with user-specific context [19, 33], voice interfaces can facilitate more natural interactions between users and their devices.

In our paper, we seek to achieve this same goal by supporting a widely used speech phenomenon known as speech repair. Allowing voice interfaces to correctly interpret natural speech disfluencies would allow them to improve their general usability outside of user-specific contexts.

2.3 Approaches to Speech Repair

While voice assistants such as Google Assistant allow developers to implement error handling by manually implementing explicit error states [1], speech repair detection is aimed at detecting and resolving such occurrences in a more general case. For example, design guidelines for some conversational agents recommend that unique error handling should be implemented for each turn, handling cases of missing or misinterpreted user input [1]. This approach allows designers to incorporate a large degree of customization for facilitating different types of error recovery (*e.g.,* no input, no match, and system errors), but it requires that additional "fallback handlers" (*i.e.,* error states) be created for each actionable intent in the agent. On the other hand, a speech repair layer could automatically resolve more general errors, removing the need for manual specification and handling.

The concept of speech repair has been widely studied in the context of speech disfluency theory [22, 37]. In particular, earlier research has examined the structure of such phenomena as a way to identify discourse markers and preprocess speech data for later analysis. While the terminologies used to describe the components of speech repair differ, researchers tend to agree on the presence of several regions [21, 22, 37].

- Reparandum - the segment of speech that should be corrected
- Interruption point - user interrupts speech
- Editing terms (Interregnum) - signifies the end of the reparandum and beginning of the speech repair
- Alteration (Repair) - segment of speech containing corrected material for reparandum

Early detection mechanisms sought to label individual parts of utterances, also called sequence tagging. Heeman and Allen proposed an algorithm to detect and fix speech repairs without syntatic or semantic knowledge of the full utterance. It instead processed words one by one to dynamically create a repair pattern, using a set of constraints for determining valid word correspondences. This was also combined with a Markov model for determining part-of-speech and judging potential repairs from the pattern builder [21]. The two also proposed integrating identification of discourse markers and speech repairs with speech recognition language models [22]. In addition to sequence tagging, another technique made use of a noisy channel model to compare the reparandum to the alteration as a means of detecting disfluency [45].

More recent data-driven approaches have used various neural models to detect and repair disfluencies [23, 24, 44]. Some of the advantages over previous techniques are lower latency, better output stability, and improved handling of more complex repair structures. Using an Elman Recurrent Neural Network (RNN) proved to be comparable to state-of-the-art methods, even without careful feature engineering [23]. Additionally, a long short-term memory (LSTM) neural network can help make up for a noisy channel model's inability to represent complicated language structures [24]. Previous sequence labelling/tagging techniques also fail to effectively capture long-distance dependencies as well as proper grammar in the utterance. However, treating disfluencies as a sequence-to-sequence problem and taking a neural machine translation approach have become popular in overcoming these challenges [13, 42].

While there have been many advances in disfluency detection and correction, these techniques have only been evaluated and applied to speech transcription corpora like the Switchboard dataset. These datasets primarily consist of conversational transcripts and do not closely resemble command-based interactions common with virtual assistants. Speech repair, as a system and interaction, has not been applied nor assessed in speech interfaces like voice assistants.

3 SCRATCHTHAT

ScratchThat supports query correction through *corrective clauses*, which serve to edit or correct portions of the original input.

In ScratchThat, we assume the presence of two clauses. The first clause contains the user's original query, and the second clause is the corrective clause, which contains language to modify or correct a parameter in the query. Depending on the type of interaction (Table 1), these parameters can be detected in different ways (Section 3.1).

This structure loosely follows the model of speech repair in spoken dialogue, where speakers begin a sentence before they are sure what they want to say [22]; however, there are some notable differences.

- Unlike the traditional speech repair model, we do not expect speech repair structures to occur in the same sentence (Section 2.3), only the presence of the the original query clause and the corrective clause. This allows our system to be used in a variety of use-cases (Section 3.1).
- We ignore the possibility where the user performs a speech repair in the middle of a sentence and continues on after the alteration has ended (*e.g.,* Can you send *Alice, I mean Bob*, this picture?).
- ScratchThat supports alterations to multiple reparanda in the original query, where the order of the alterations are not necessarily in the same order.

While some restrictions prevent the interaction from being as natural as spoken dialogue, we believe that this model is more appropriate than the traditional model (Section 2.3) which makes a number of assumptions such as fixed placement of edit terms and changed parameters. From our study results (Table 5), we find that such assumptions would not hold true for a significant percentage of collected responses. Moreover, prompts with multiple parameters could elicit non-immediate or out-of-order repairs. In addition, we seek to support flexible speech interactions with a variety of devices with and without screens, which entail differing methods of usage and feedback.

In the following sections, we provide more information on how our system ScratchThat can support different use-cases and different steps of the ScratchThat algorithm. We build our implementation using freely available NLP and machine learning software packages capable of running at latencies of under 5 seconds on a laptop computer with a quad-core processor and 16 GB of RAM. In real-world deployment scenarios, the algorithm can either be optimized to run on a phone using compressed models or be run as a cloud-based service. In the latter case, the algorithm's performance, specifically embedding inference, can be greatly improved using a graphics processing unit (GPU) or specialized tensor processing unit (TPU) [25]. This would allow for near immediate processing time and offload device computation to minimize battery impact.

3.1 Repair Interactions

Modern conversational agents often provide output that extends beyond pure text-only responses. Consumer virtual assistants, such as the Google Assistant, allow developers to include multi-modal functionality [2] which

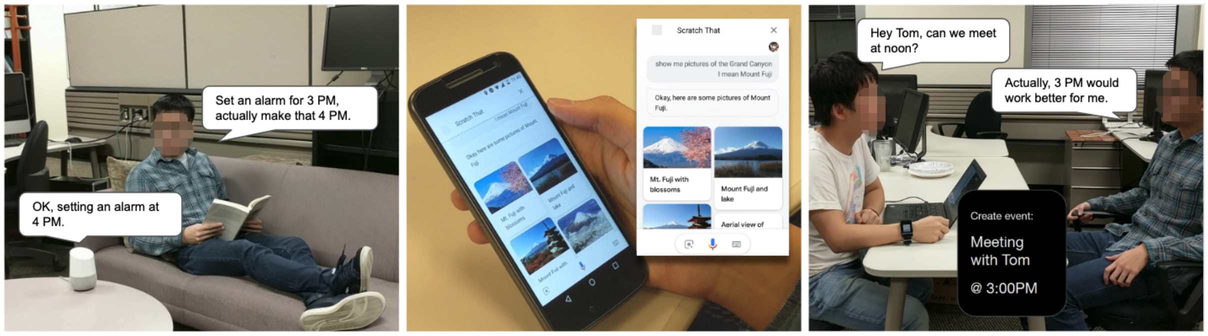

Set an alarm for 3 PM, actually make that 4 PM.
OK, setting an alarm at 4 PM.
Scratch That
show me pictures of the Grand Canyon I mean Mount Fuji
Okay, here are some pictures of Mount Fuji.
Mt. Fuji with blossoms
Mount Fuji and lake
Aerial view of
Hey Tom, can we meet at noon?
Actually, 3 PM would work better for me.
Create event:
Meeting with Tom
@ 3:00PM

Fig. 1. Example applications of the ScratchThat system. (Left) A user interacts with a smart speaker. (Center) A user searches for images on a smartphone. (Right) A dual-purpose speech agent is able to correctly intercept meeting details from a natural conversation between two people.

Table 1. Characteristics of Voice Assistant Interactions

Type	Description	Assumptions	Devices
Conversational	Text or spoken dialog responses	Screen not present Turn-based interaction Feedback each turn	Smart speakers
Visual	Responses presented with visual media	Screen is present Turn-based interaction Immediate feedback	Smartphone assistants
Passive	Using third-party conversation as input	Screen not present No direct interaction No feedback	Varies

has implications on the supported conversational interactions and device hardware (Table 1). Below, we elaborate on the integration and use-cases for conversational, visual, and passive interactions for voice assistants.

3.1.1 Conversational Components. Conversational components of voice assistant interactions are usually supported by turn-based speech which can serve to prompt or provide feedback to the user. ScratchThat can be implemented as a layer on top of voice assistants that automatically pass resolved queries to underlying commands without built-in support for error handling.

It is relatively simple to retrieve both the original query and correction clause from two separate turns. However, due to the relatively long time required to use speech to present information, it is likely for the speech repair to occur in the same turn as the original query (*e.g.,* Can you send Alice this picture, *I mean Bob?*). In these scenarios, a list of edit terms or a repair keyword can be used to segment the input into the query clause and corrective clause.

3.1.2 Visual Components. Voice assistants on devices with screens enable multi-modal interactions incorporating visual components. In addition to supporting different response types, voice assistant GUIs often provide other features such as real-time transcription feedback and alternate ways of interacting with the assistant other than speech.

A major difference between visual and conversational components is the greater speed at which information can presented. In addition to supporting conversation-based repairs, it is also feasible for visual assistants to display a confirmation screen, which can quickly be confirmed or used to initiate a speech repair. This clear separation between the original query and corrective clause allows users to be more flexible in constructing natural repair phrases, as the segmentation would not depend on detecting the presence of specific edit terms or keywords.

3.1.3 Passive Interactions. Outside the scope of common user-assistant interactions, it is possible to improve passive speech interactions such as dual-purpose speech using speech repair.

Lyons et al. [28] demonstrated the possibility of leveraging patterns in everyday speech as hooks for automatically recording important bits of information vocalized throughout the day. For example, it is common courtesy to echo back a phone number as it is being given to you. An always-on microphone might be able to detect this phenomenon and automatically create a contact card with the phone number.

This presents some challenges for our system due to the absence of feedback and the possible presence of repairs in the middle of sentences, which would violate some of our assumptions. However, we believe ScratchThat can still improve the accuracy of such systems by detecting and resolving speech repair phrases.

3.2 Parameter Identification

ScratchThat automatically identifies replaceable entities in the speech query for use as command parameters. We employ an algorithm that does not require labelling of parameter slots or expected command arguments and instead groups words together based on part-of-speech (POS). Because of this, ScratchThat can be used "out of the box" with most virtual assistants or information retrieval commands.

This approach detects most common noun phrases but may not correctly identify certain named entities or proper nouns. We accommodate this by augmenting the parameter identification with named entity recognition (NER) models and a user-specific list of known entities (*e.g.,* list of users' songs, contacts).

3.2.1 Chunking. Chunking, also known as shallow parsing, involves grouping words in the input query into higher order segments that represent different entities or actions. In particular, noun chunks can often be used to fill subject and object positions in a sentence [7], and we use them as possible query parameters or replacement candidates. While there are many methods to identify chunks through parsing grammars or machine learning, we employ a simple algorithm which groups words by their POS tags using regular expressions.

The POS tags are computed using the Stanford CoreNLP POS tagger [39], which outputs the Penn Treebank tag set. The CoreNLP POS tagger is trained mostly on articles, documents, and other declarative, written language and is not necessarily optimal for tagging conversational speech commands (Figure 2). However, we find its performance is sufficient for the purpose of noun phrase extraction.

3.2.2 Named Entities. ScratchThat can use a combination of automatic named entity recognition and values from a user-specific entity list to revise incorrectly chunked entities (*e.g.,* a title such as *Harry Potter and the Sorcerer's Stone* would be chunked as two entities) and to detect additional entities (*e.g.,* time descriptors). We leverage the a pretrained NER tagger available in the Stanford CoreNLP library [16] to generate tags for a variety of entities (Location, Person, Organization, Money, Percent, Date, Time). This model was trained on the MUC-7 dataset, corpus of news articles from the 1990s [12].

While the NER model detects time descriptors well and can recognize most common names, it is unable to detect titles of recently released media or some names. In real-world use cases, the accuracy of NER for some voice assistants can be further improved by requesting relevant data for certain usage scenarios. For example, the names on a user's contact list can aid a smartphone-based virtual assistant in detecting the message recipient

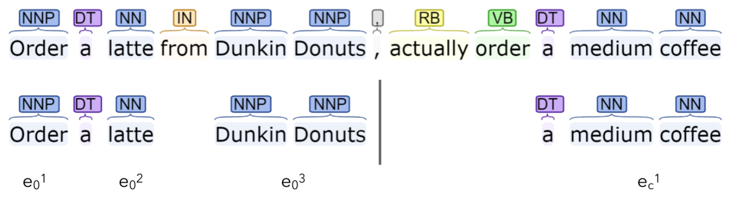

NNP DT NN IN NNP NNP , RB VB DT NN NN
Order a latte from Dunkin Donuts , actually order a medium coffee
NNP DT NN NNP NNP DT NN NN
Order a latte Dunkin Donuts a medium coffee
e0^1 e0^2 e0^3 ec^1

Fig. 2. Entity extraction from example input by chunking noun phrases. Part of speech labelling is not entirely correct, due to the loose nature of conversational grammar. However, all relevant entities (true positives) are detected.

parameter of an email application. Similarly, a list of songs in a user's media library can be used with a smart speaker's music playing functionality.

3.3 Query Correction

Our algorithm assumes that the original mistaken entity will be semantically and syntactically similar to the intended replacement. While the assumption may not always hold, such as in the case where the speech-to-text (STT) provides a similar-sounding or homophonic mistranscription, we believe these assumptions are generally valid for the reasons below.

- The language model in modern STT minimizes the probability that mistransciptions due to similar-sounding but semantically different words will occur.
- We investigate primarily cases of query correction where the overall intent of the action remains unchanged, but the user changes his/her mind about one or more parameters. Since each parameter represents a certain type of entity (*e.g.*, date, name, place, food), valid entities for each slot are generally more semantically similar than invalid entities.
- The overall syntactic structure of the query reveals parameters' relations to other entities and actions. Valid inputs to a parameter will generally have similar relationships to the rest of the query because the overall outcome and expected arguments of the command stay constant.

Thus, we design a similarity measure that is able to determine the likelihood that a given replacement candidate is valid for the input and correction pair. This similarity measure is scored using both the semantic, $similarity_{sem}$, and syntactic similarity, $similarity_{syn}$, of the inputs.

We use these similarity scores to construct a cost function that is used to find the optimal matching between entities in the corrective clause and parameters in the query.

3.3.1 *Semantic Similarity.*

We define a semantic similarity measure, $similarity_{sem}$ that compares the similarity of two entity inputs by comparing their text embeddings. Text embeddings encode text input into vector representations that are useful for natural language processing tasks such as classification and semantic similarity.

For the semantic similarity task, we use the Universal Sentence Encoder [9], for which there is a pretrained model readily available online. We select this encoder due to its ability to accept and generate representations for entity chunks, which are often greater than word length.

The semantic similarity measure is computed with the commonly used cosine similarity metric for natural language processing tasks. Due to the Universal Sentence Encoder's ability to accept greater-than-word length text, it is possible to compute the embedding similarity between both the entities themselves or candidate queries (modified versions of the input query where some of the original entities are replaced with correction candidates).

	A Drink	McDonald's	A burger
A Drink	1	0.34	0.53
McDonald's	0.34	1	0.74
A burger	0.53	0.74	1

Entity Similarity

	A Drink	McDonald's	A burger
A Drink	1	0.91	0.84
McDonald's	0.91	1	0.8
A burger	0.84	0.8	1

Sentence Similarity

Fig. 3. Semantic similarity scores when comparing entity similarity and sentence similarity for the input query "Order *a drink* from *McDonald's*" and corrective clause "actually make that *a burger*." Entity similarity is computed as the $similarity_{sem}$ between the entities themselves while sentence similarity is computed as the $similarity_{sem}$ between the original input query and the input query with the entity "a drink" substituted for the replacement candidate.

While there is no limit on the length of text that the encoder can accept, longer text inputs lead to more "diluted" embeddings, where it becomes more difficult to distinguish between candidate queries (Figure 3).

However, using the entity similarity to measure query similarity may cause an interesting problem, where the corrective entities are mis-assigned due to strong associations between parameters of different types. Figure 3 shows the entity similarity and sentence similarity matrices of the input query "Order *a drink* from *McDonald's*" and the corrective clause "actually make that *a burger*."

The entity similarity matrix shows that *a burger* has a stronger association with *McDonald's*, which would cause the resulting query to become "Order *a drink* from *a burger*." This is due to the strong association between McDonald's and selling burgers, which overshadows the relationship between burgers and drinks as food items that are able to be ordered from a restaurant.

On the other hand, the sentence similarity matrix, which embeds the entire candidate query, shows that the entities *a burger* and *a drink* have a higher similarity score than *a burger* and *McDonald's*. This would result in the intended query "Order *a burger* from *McDonald's*."

Nevertheless, it is difficult to design an algorithm that is guaranteed to overcome the problem of extra associations between two entities. This is in part due to the command-agnostic nature of ScratchThat, where no knowledge of the command structure and expected arguments is needed. It is possible to provide knowledge of parameter types, constraining the possible candidates and further decreasing the likelihood of incorrect replacement. In the next section, we describe a syntactic similarity measure that resolves many cases where entity confusion is possible, including the example described.

3.3.2 Syntactic Similarity. The syntactic similarity score, $similarity_{syn}$, compares entities' relations to other parts of the sentence. Given the command-agnostic nature of ScratchThat, where no information about command structure or expected arguments are given, it is especially important to capture the sentence structure of the query, as this can be used to infer entity type and correct errors.

To compute the syntactic similarity, we use the directed edges of a dependency tree to model entity relations. The dependency tree is computed with the Stanford CoreNLP neural dependency parser [11], a transition-based neural dependency parser that produces projective trees and POS tags.

Figure 4 shows the output of the CoreNLP dependency parser. Note that although the parser produced an incorrect POS tagging for the clauses (*Order* is incorrectly tagged as a singular proper noun), the dependency tree is correct. The **dobj** edge from *Order* to *latte* indicates a verb-direct object relationship between the two words, correcting the POS tag.

The syntactic similarity of two entities is computed as the Jaccard similarity between the entities' dependency edges after replacement. The Jaccard similarity is a commonly used similarity measure for finite sets and is defined as the size of the intersection divided by the size of the union. For each entity, we construct a set of incoming and outgoing edges from words not in the entity itself.

Figure 4 shows the syntax similarity score computed between two candidate replacements for the input query "Order *a latte* from *Dunkin Donuts*" and correction clause "actually order *a medium coffee*." It is interesting to note that another replacement, "Order *a latte* from *a medium coffee*" would also result in perfect semantic similarity, but achieves a lower semantic similarity score.

Thus, we show that the semantic similarity measure, $similarity_{sem}$, and the syntactic similarity measure, $similarity_{syn}$, complement each other and are both required for correctly evaluating replacement candidates.

3.3.3 Entity Matching. We formulate the final entity matching problem as a minimum cost assignment problem between the set of entities in the original query, $\{e_o^1, e_o^2, \ldots, e_o^n\}$, and the set of entities in the corrective clause, $\{e_c^1, e_c^2, \ldots, e_c^n\}$ (Figure 5).

$$cost = 1 - (\alpha \cdot similarity_{sem} + (1 - \alpha) \cdot similarity_{syn}) \tag{1}$$

The linear sum assignment optimizer included in SciPy, which implements the Hungarian algorithm [27], is used to compute the optimal assignment. This assignment assumes every entity in the correction clause will be matched to a target, but due to false positives in the chunking algorithm and the presence of irrelevant entities, we introduce an additional thresholding step. The threshold ensures that a replacement is made only if the replacement cost is less than an empirically set threshold ($thresh_{max} = 0.6$).

4 EVALUATION

4.1 Correction Clause Elicitation

We created command templates (Table 2) from an easily-found online list of supported commands for the Alexa virtual assistant [3]. Although we chose to evaluate Alexa-supported commands for our study, we intentionally chose very common commands that have similar counterparts on other virtual assistants as well so that the results from our studies can be more generalizable. Nevertheless, the scope of our studies is mostly limited to the use-case of an Alexa Conversational Voice Assistant (Table 1).

In particular, we selected all productivity and information-retrieval commands that would be used in a workplace scenario to more closely model continued, in-situ usage of a voice assistant in such a situation rather than fragmented use across unrelated contexts. This was done so that the real-world use-case of the prompts would more closely match pace of this study and the interaction comparison study, where users respond in quick succession.

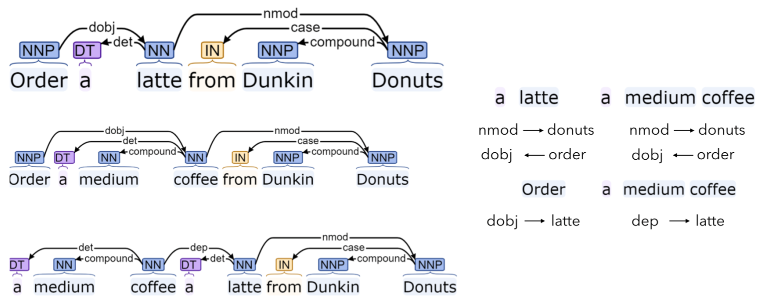
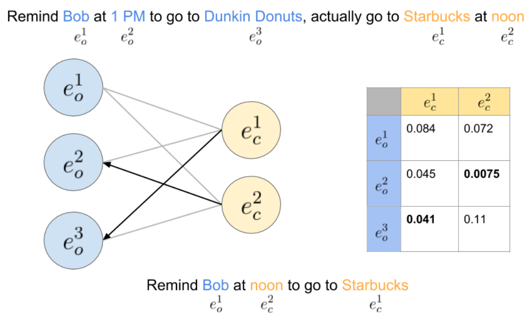


Fig. 4. Syntactic similarity score using dependency graph edges. This score is able to eliminate incorrectly identified entities in the chunking phase by identifying syntactic incompatibility. An example of a good match and bad match are shown. In this case, $similarity_{syn}$(a_latte, a_medium_coffee) = 1 and $similarity_{syn}$(Order, a_medium_coffee) = 0.


Remind Bob at 1 PM to go to Dunkin Donuts, actually go to Starbucks at noon
e_o^1 e_o^2 e_o^3 e_c^1 e_c^2
e_o^1
e_o^2
e_o^3
e_c^1
e_c^2
Remind Bob at noon to go to Starbucks
e_o^1 e_c^2 e_c^1

	e_c^1	e_c^2
e_o^1	0.084	0.072
e_o^2	0.045	**0.0075**
e_o^3	**0.041**	0.11

Fig. 5. Entity matching and replacement formulated as a minimum cost assignment problem. Here, an example of the entity matching is shown with multiple parameters and replacement candidates. Table represent costs between replacements, and the assignment with the least cost is used to generate the output.

Table 2. Commands and Parameter Types

Commands	Parameter Types
Financial Markets (3)	Stock Ticker, Date
Travel Information (4)	City, Country
Communication (3)	Person Name
Calendar (4)	Event, Date
Reminders (3)	Date, Person Name
Clock (3)	Time, Duration, Date

Table 3. Parameters Types and Examples

Parameter Types	Examples
Stock Ticker	GOOG
Date	May 6th, next Tuesday
City	London
Country	Australia
Person Name	Alice
Event	the hackathon
Time	3:45 PM, noon
Duration	5 minutes, 1 hour 10 minutes

Table 3 shows command templates and parameter types used for generating prompts. Each parameter type contained at least 10 different possible values meant to test a diverse set of representations. For example, the Date parameter included both absolute (May 6th) and relative (next Tuesday, tomorrow) dates.

Using a prompt generator that randomly selects from the available command templates and parameters, we generated two datasets, one for use with an online Amazon Mechanical Turk (MTurk) elicitation study and one for later use with an interaction comparison user study. Each dataset contained a list of input queries, randomly selected parameter modifications, and “ground truth” replacement queries. The MTurk dataset contained 40 prompts, each consisting of an initial query and correction. For the user study, we generated a separate dataset consisting of 10 sets of 20 prompts to match our intended study design of asking 10 participants to engage in a voice-assistant usage session of 20 prompts. Although we generated a new set of prompts to further increase the diversity of the dataset, these prompts were generated using the same command templates (Table 2) and possible parameters (Table 3) as the MTurk dataset and are directly comparable.

While the primary purpose of the elicitation study was to characterize natural forms of command correction with regard to edit term placement, the MTurk study was also used to inform the design of ScratchThat and construct a large dataset of query repair examples for offline evaluation of the repair algorithm. We created an online survey that randomly chose 5 prompts from the MTurk question bank every time it was started. Each of the 5 prompts included an input query and a parameter correction. Although the online survey was presented in a text format due to limitations of the platform, survey-takers was instructed to enter phrases that they would verbalize aloud. Furthermore, we asked participants to construct a repair phrase that they would use with a virtual assistant that had human-level understanding of speech correction (*i.e.,* “a virtual assistant that can understand you as well as a human being could."). Our specific phrasing is aimed at producing repair phrases that naturally occur in human conversations but roughly adhere to the constraints of virtual assistant queries (*e.g.,* fits inside a single turn, fewer occurrences of other speech disfluencies and filler words, etc...). The survey was posted on Amazon’s Mechanical Turk platform as a masters-only task for 3 days and was completed 120 times, resulting in 600 responses. 27 responses were identified as incorrect or not following instructions and removed. The final dataset consisted of 573 responses from 118 users.

Table 4 shows the most frequently used edit terms for the MTurk dataset. Although the MTurk dataset was collected using an online survey, common edit terms such as “I meant” and “sorry” represent phrases associated with spoken English.

In addition, we examined the positioning of the edit terms within the responses (Table 5). We found that out of 573 total responses, 49% of responses had edit terms in the beginning half of the phrase, 11% had edit terms at the last half of the phrase, and 40% did not contain any edit terms at all. This suggests that speech interface users interact with virtual assistants differently from normal conversational contexts and confirm some of the assumptions made in Section 3. Thus, the traditional speech repair model discussed in the Section 2.3 cannot be relied on too heavily for accurate command repair. On the other hand, the ScratchThat algorithm enables more flexible speech repair, allowing non-immediate, out-of-order, and multiple entity replacements at once.

4.2 Interaction Comparison

In order to evaluate our ScratchThat algorithm, we conducted an experiment to assess the interaction speed as well as the task load of our technique compared to other speech repair methods.

4.2.1 Conditions. We implemented four different methods of retroactively correcting queries which, aside from our ScratchThat algorithm, can already be found implemented in speech interfaces like commercial dictation systems [4–6]. They are “repeat”, “delete”, “replace”, and “ScratchThat”. We describe each condition in more detail below and provide examples for query correction.

Consider the scenario in which a user has made the query: “Set a repeating alarm every Tuesday at 3:45 PM” and would like to repair the query so that the alarm is set every Wednesday instead of every Tuesday.

- Repeating the entire query again (*e.g.,* “Set a repeating alarm every Wednesday at 3:45 PM”)
- Saying “delete” to simulate a backspace on the query buffer (*e.g.,* “*delete delete delete* Wednesday at 3:45 PM”)
- Saying “replace” to modify the word(s) in the query with new word(s) (*e.g.,* “*Replace* Tuesday *with* Wednesday”)
- Using ScratchThat by adding a natural correction clause (*e.g.,* “*Actually* make that Wednesday”)

4.2.2 Tasks. For the study’s data collection, we built an interactive prompting application that resembles a voice assistant GUI (Figure 1). The prompting application used the voice recognition functionality, as part of the W3C Web Accessibility Initiative standard, available in Google Chrome version 70. The study participant was given instructions through the prompting application and followed the instructions by interacting with the application using voice commands. Each prompt was given to the participant included an original input query and a parameter correction. The participant repeated aloud the input query word for word. Then, using the given parameter correction and one of the four repair methods, the participant said the voice command to modify their original input query.

For the previously described example of setting a repeating alarm every Wednesday instead of Tuesday, we provide the series of prompts that are sent to the participant.

Prompt: "Set a repeating alarm every Tuesday at 3:45 PM."
Participant: "Set a repeating alarm every Tuesday at 3:45 PM."
Prompt: "Tuesday $\rightarrow$ Wednesday"
Participant: *Participant says correction phrase for current condition.*

Note that the parameter correction prompt is shown to the user using an arrow symbol. This is meant to discourage the user from simply reading the prompt and to force them to construct their own natural language responses that follow the given condition's correction scheme.

Compared to the other three conditions, "ScratchThat" inherently allows participant to be more flexible in their response to the parameter correction prompt, which also introduces some ambiguity for when a repair phrase is considered "correct". For the interaction comparison user study, validity of the response is determined by a Wizard-of-Oz experimenter in-situ. Later, in our evaluation on ScratchThat's correction accuracy (Section 4.3), we definitively evaluate the algorithm's ability to detect and correct repair phrases with varying edit terms.

4.2.3 Study Design and Procedure. We recruited 10 participants (ages 21-27, mean 23.6, 8 males and 2 females) to evaluate our system. Participants were recruited from our university campus using convenience sampling and resulted in a population with diverse English speaking fluency that more realistically models the real-world user base of voice assistants. Only three participants were native American English speakers (*i.e.,* spent a significant part of their lives in America), and the remaining had varying degrees of accent in their speech. However, all participants had sufficient English speaking ability to be enrolled in a United States university. Based on our anecdotal observations, differences in English fluency mostly affected the accuracy of the speech transcription program rather than speech repair phrases used.

In total, the study lasted around one hour and was composed of four sessions, each meant to evaluate one of the conditions. Before the beginning of the study, participants were read the description and purpose of the study and were asked to sign a consent form. Upon completion of the study, participants were paid $10. We used a Latin square to partially counterbalance the four condition methods per user. Each session involved one condition method and 20 prompts, and participants completed the four sessions in the order determined by the Latin square.

Participants sat in front of a laptop computer displaying the earlier described prompting application. Each prompt was displayed to the participant, who was given instructions specific to each task (Section 4.2.2). The study was conducted with a Wizard of Oz, who determined the correctness of the transcribed speech of the user's voice commands for each prompt. In some cases, accented speech that was mistranscribed by STT was forgiven at the Wizard's discretion. At the start of the study, participants learned how to use the condition methods and the prompting application by completing a practice session including five prompts for each of the four conditions. The experimental sessions followed afterwards. The set of 20 prompts for each condition were taken from the user study dataset generated in Section 4.1. We also used the NASA TLX survey to evaluate the comfort and workload on the user for each of the experimental conditions.

For each participant, we recorded the amount of time taken to complete each prompt (both the time taken to say the prompt and the time taken by the recognizer) as well as the transcription of their voice as they read the prompts. From this information, we evaluate the interactions in terms of speed and comfort that ScratchThat enables, as well the accuracy of our algorithm. We analyzed the interaction speed and task load data using paired t-tests adjusted with Bonferroni correction.

4.2.4 User Performance. Our first evaluation metric was to compare the time taken to complete a task. We hypothesize that the more natural it is to execute a task, the faster it should be. Our results show that ScratchThat, on average, was the fastest method. In fact, it was significantly faster than "repeat" ($p < 0.05$) and "replace" ($p < 0.01$), and "delete" ($p < 0.01$).

Table 4. Top Edit Terms for MTurk Dataset (Left) and User Study Dataset (Right)

Edit Term	Frequency	Edit Term	Frequency
change	28%	change	27%
I meant	12%	actually	24%
instead	12%	make that	14%
actually	5%	I meant	13%
sorry	4%	make it	12%

Table 5. Edit Term Location for MTurk Dataset (Left) and User Study Dataset (Right)

Location	Frequency	Location	Frequency
Beginning	49%	Beginning	69%
End	11%	End	3%
None	40%	None	28%

After completing all of the tasks, we asked participants to fill out a TLX survey in order to evaluate the perceived load required. We found that, on average, participants preferred "ScratchThat" in terms of effort and frustration; "repeat" for performance and mental demand; and "replace" for temporal and physical demand. On the other hand, the "delete" task required the highest load across all of the tasks due to the user having to count how many words to delete, with significance when compared to ScratchThat on mental demand ($p < 0.05$), frustration ($p < 0.01$), and effort ($p < 0.01$). This finding is encouraging since the "delete" method is used in currently existing commercial systems, showing that we could potentially make such systems much less strenuous to use.

4.3 Correction Accuracy

In addition to exploring natural methods of speech repair and evaluating speech repair interactions, we conducted offline evaluations of the ScratchThat algorithm described in Section 3. Specifically, we aimed to assess the

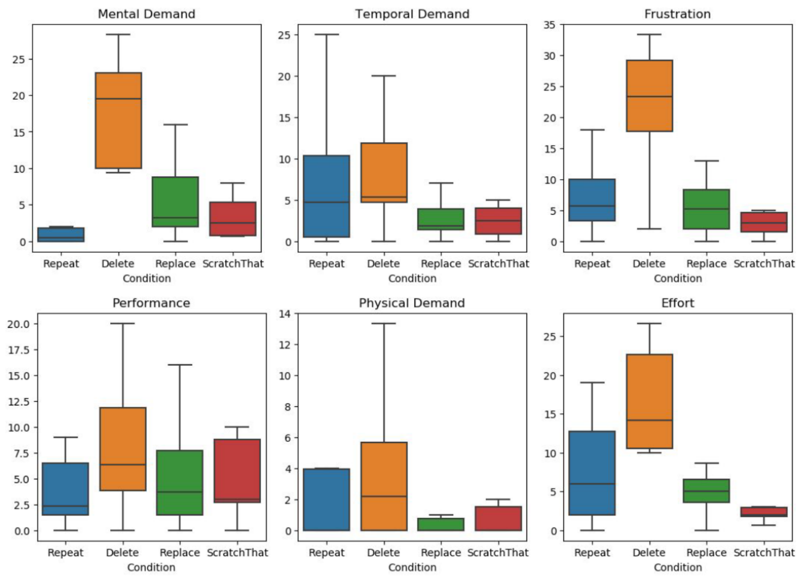


Fig. 6. TLX survey scores is shown for 6 dimensions. A lower score, indicating less load, is desirable. On average, ScratchThat had a lower effort and frustration while Repeat was preferred for performance and mental demand.

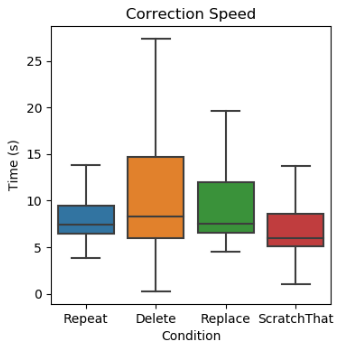


Fig. 7. Average time taken by each correction method. Our system, ScratchThat, was significantly faster than others.

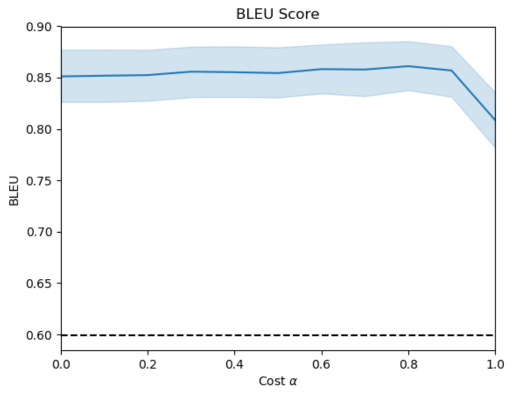


Fig. 8. BLEU score vs cost α for MTurk dataset. Dotted line shows BLEU score for unmodified prompts.

Table 6. BLEU score and correction accuracy. BLEU score of unmodified prompts are also shown as reference. Fleiss's Kappa (κ) is also shown.

	MTurk		User Study	
	Prompt	Corrected	Prompt	Corrected
BLEU	0.60 ± 0.17	0.86 ± 0.31	0.65 ± 0.16	0.94 ± 0.26
Acc.	-	79% κ = 0.92	-	77% κ = 0.84

validity of the assumptions made in Section 3.3 and the accuracy of our system in the context of speech command data rather than regular conversational data used by previous work [13, 21–24, 42, 45].

We performed an automated machine translation and manual human rating evaluation on both the MTurk (573 responses) and User Study (190 responses) datasets. Both datasets were first preprocessed to fix minor misspellings and mistranscriptions. In addition, during the user study, some prompts (*e.g.,* "What was the stock price for GOOG last Tuesday?") contained company tickers as parameters. Many participants chose to say the companies associated with the stock ticker rather than reading the tickers themselves (*e.g.,* "What was the stock price for *Google* last Tuesday?"). We programmatically replace all occurrences with the company names for consistency.

BLEU is a method for evaluating the quality of a machine translation by computing the geometric mean of n-gram precisions [10]. Previous work has used the BLEU score metric for automated evaluation of speech repair systems on large corpora [13]. However, because speech repair generally replaces only limited portions of the sentence and most of the query remains the same, the BLEU score gives an over-optimistic score for the quality of the repair correction. In fact, Table 6 and Figure 8 shows that the BLEU scores for the unmodified prompts are already above 0.6. Thus, while we use an automatically calculated BLEU score for parameter tuning, we conduct our final evaluation with human raters. We use the sentence-level BLEU metric with the Smoothing 7 smoothing function described by Chen and Cherry [10].

Figure 8 shows the result of the α parameter tuning. The maximum BLEU score is obtained at $\alpha = 0.8$ ($m_{0.8} = 0.86$) on the MTurk dataset, which is used as the α for both datasets' scores in Table 6.

We also show the correction accuracy of ScratchThat on both datasets. The algorithm is run on the original prompts and corresponding correction clauses, and the output is given to 4 human raters. The raters were instructed to mark the response as correct if the output would produce the correct action if given to a virtual assistant. In addition, raters were told to ignore the stop words that are present in the transcribed user study dataset.

Table 6 shows the resulting accuracies for the MTurk dataset ($m_{mturk} = 79\%$) and the User Study dataset ($m_{study} = 77\%$). Although the corrected BLEU score and BLEU score increase was higher for the User Study dataset, we find the ScratchThat algorithm performs relatively consistently across both datasets.

5 LIMITATIONS & FUTURE WORK

Throughout this paper, we investigated both interaction methods for command correction and the performance of the ScratchThat algorithm. We propose next steps for command correction as an interaction and the ScratchThat algorithm.

In our MTurk elicitation study, we collected a large number of command correction examples ($n_{examples} = 573$) from online crowdworkers ($n_{responses} = 120$). While the nature of the online study limited participant responses to a textual form, we were able to collect a dataset of imagined speech responses. Our results (Tables 4 & 5) showed that speech repair with virtual assistant interactions was not well described by traditional conversational speech repair models described in Section 2.3. Specifically, the editing terms that are commonly used to segment the reparanda from alterations does not always appear in front of the alteration phrase, making it difficult to identify repair segments within a single sentence. If a traditional model of speech repair was assumed, then at least 30% of the prompts in the user study would be misinterpreted. For future work, we seek to construct a system that can consistently and accurately identify speech repair regions. This would allow more natural user interactions, supporting mid-sentence repairs and different types of disfluencies.

We also described the ScratchThat algorithm, which leveraged several pretrained NLP models for POS chunking, NER, text embedding, and dependency parsing. Although these models were trained primarily using text documents and were not tuned to handle speech data, we show that our approach is able to produce accurate repairs when evaluated by humans and machines (Table 6). For future work, we seek to further improve repair correction by training on a corpus of conversation data or speech commands.

Finally, our Interaction Comparison study was not representative of all types of voice assistant interactions (Table 1). Specifically, we evaluated ScratchThat in the context of a common voice-based conversational assistant (*i.e.*, Alexa) for productivity and information-retrieval commands. There remains significant opportunity for evaluating speech repair in broader use-cases, with a greater number of participants and types of speech commands. In particular, the accurate detection and processing of speech repair in passive interaction scenarios (Section 3.1.3) may differ significantly, due to the presence of free-form conversation and lack of virtual assistant feedback to the user.

6 CONCLUSION

In this paper, we presented ScratchThat, a method for supporting command-agnostic speech repair in voice-driven assistants. Our work was motivated by the increasing popularity of voice interfaces and virtual assistants and the need to improve their usability by enabling command correction.

We first described the ScratchThat algorithm that supports command-agnostic speech repair by automatically inferring query parameters and computing a novel heuristic for replacement assignment. Our heuristic uses the semantic and syntactic similarity between entities to support a more generalized repair model capable of inferring multiple, non-immediate corrections.

We conducted a series of evaluations to elicit natural forms of command repair, compared the interaction speed and task load of the system to existing voice-based correction methods, and assessed the accuracy of the ScratchThat algorithm on two datasets (over 700 prompts in total). Our results show faster interaction times for speech repair-based command correction and 77%-79% correction accuracy.

REFERENCES

[1] [n. d.]. Actions on Google Errors - Conversational components. https://designguidelines.withgoogle.com/conversation/conversational-components/errors.html. Accessed: 2019-01-25.

[2] [n. d.]. Actions on Google Scale your design - Conversation design process - Conversation design. https://designguidelines.withgoogle.com/conversation/conversation-design-process/scale-your-design.html. Accessed: 2019-01-26.

[3] [n. d.]. Alexa.bio The Living List of Alexa Commands. Accessed 2018-11-15.

[4] [n. d.]. Dictanote How to setup Voice Commands? https://support.dictanote.co/hc/en-us/articles/115002811807-How-to-setup-Voice-Commands-. Accessed: 2019-04-11.

[5] [n. d.]. Dragon Inserting, replacing, and deleting text. https://www.nuance.com/products/help/dragon/dragon-for-pc/enx/professionalgroup/main/Content/WorkingWithText/inserting_replacing_and_deleting_text.htm. Accessed: 2019-04-11.

[6] [n. d.]. Dragon Revising text. https://www.nuance.com/products/help/dragon/dragon-for-mac/enx/Content/Correction/RevisingText.htm. Accessed: 2019-04-11.

[7] Steven P Abney. 1991. Parsing by chunks. In *Principle-based parsing*. Springer, 257–278.

[8] Frank Bentley, Chris Luvogt, Max Silverman, Rushani Wirasinghe, Brooke White, and Danielle Lottrjdge. 2018. Understanding the Long-Term Use of Smart Speaker Assistants. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 3 (2018), 91.

[9] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).

[10] Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*. 362–367.

[11] Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 740–750.

[12] Nancy Chinchor. 1998. Overview of MUC-7. In *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29-May 1, 1998*.

[13] Eunah Cho, Jan Niehues, Thanh-Le Ha, and A. Waibel. 2016. Multilingual Disfluency Removal using NMT.

[14] Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José MF Moura, Devi Parikh, and Dhruv Batra. 2017. Visual dialog. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 2.

[15] Ethan Fast, Binbin Chen, Julia Mendelsohn, Jonathan Bassen, and Michael S Bernstein. 2018. Iris: A Conversational Agent for Complex Tasks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 473.

[16] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 363–370.

[17] 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*. ACM, 1617–1623.

[18] Sayan Ghosh, Mathieu Chollet, Eugene Laksana, Louis-Philippe Morency, and Stefan Scherer. 2017. Affect-lm: A neural language model for customizable affective text generation. *arXiv preprint arXiv:1704.06851* (2017).

[19] Ramanathan Guha, Vineet Gupta, Vivek Raghunathan, and Ramakrishnan Srikant. 2015. User Modeling for a Personal Assistant. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (WSDM '15)*. ACM, New York, NY, USA, 275–284. https://doi.org/10.1145/2684822.2685309

[20] Yawen Guo. 2018. *ImprovChat: An AI-enabled Dialogue Assistant Chatbot for English Language Learners (ELL)*. Ph.D. Dissertation. OCAD University.

[21] Peter Heeman and James Allen. 1994. Detecting and correcting speech repairs. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 295–302.

[22] Peter A Heeman and James F Allen. 1999. Speech repairs, intonational phrases, and discourse markers: modeling speakers' utterances in spoken dialogue. *Computational Linguistics* 25, 4 (1999), 527–571.

[23] Julian Hough and David Schlangen. 2015. Recurrent neural networks for incremental disfluency detection. In *INTERSPEECH*.

[24] Paria Jamshid Lou and Mark Johnson. 2017. Disfluency Detection using a Noisy Channel Model and a Deep Neural Language Model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 547–553. https://doi.org/10.18653/v1/P17-2087

[25] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 1–12.

[26] Boris Katz, Gary Borchardt, Sue Felshin, and Federico Mora. 2018. A Natural Language Interface for Mobile Devices. (2018).

[27] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[28] Kent Lyons, Christopher Skeels, Thad Starner, Cornelis M. Snoeck, Benjamin A. Wong, and Daniel Ashbrook. 2004. Augmenting Conversations Using Dual-purpose Speech. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 237–246. https://doi.org/10.1145/1029632.1029674

[29] Michael L Mauldin. 1994. Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In *AAAI*, Vol. 94. 16–21.

[30] 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.

[31] Daniel OâĂŹSullivan. 2009. Using an adaptive voice user interface to gain efficiencies in automated calls. *White Paper, Interactive Digital, Smithtown, USA* (2009).

[32] Aung Pyae and Tapani N. Joelsson. 2018. Investigating the Usability and User Experiences of Voice User Interface: A Case of Google Home Smart Speaker. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct (MobileHCI '18)*. ACM, New York, NY, USA, 127–131. https://doi.org/10.1145/3236112.3236130

[33] Reza Rawassizadeh, Chelsea Dobbins, Manouchehr Nourizadeh, Zahra Ghamchili, and Michael Pazzani. 2017. A natural language query interface for searching personal information on smartwatches. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 679–684.

[34] Sherry Ruan, Jacob O Wobbrock, Kenny Liou, Andrew Ng, and James A Landay. 2018. Comparing Speech and Keyboard Text Entry for Short Messages in Two Languages on Touchscreen Phones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 4 (2018), 159.

[35] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. 2016. Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models.. In *AAAI*, Vol. 16. 3776–3784.

[36] Ben Shneiderman. 2000. The Limits of Speech Recognition. *Commun. ACM* 43, 9 (Sept. 2000), 63–65. https://doi.org/10.1145/348941.348990

[37] Elizabeth Ellen Shriberg. 1994. *Preliminaries to a theory of speech disfluencies*. Ph.D. Dissertation. Citeseer.

[38] Shimpei Soda, Masahide Nakamura, Shinsuke Matsumoto, Shintaro Izumi, Hiroshi Kawaguchi, and Masahiko Yoshimoto. 2012. Implementing virtual agent as an interface for smart home voice control. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, Vol. 1. IEEE, 342–345.

[39] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 173–180.

[40] Oriol Vinyals and Quoc Le. 2015. A neural conversational model. *arXiv preprint arXiv:1506.05869* (2015).

[41] Richard S Wallace. 2009. The anatomy of ALICE. In *Parsing the Turing Test*. Springer, 181–210.

[42] Shaolei Wang, Wanxiang Che, and Ting Liu. 2016. A Neural Attention Model for Disfluency Detection. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 278–287.

[43] Jason Wu, Sayan Ghosh, Mathieu Chollet, Steven Ly, Sharon Mozgai, and Stefan Scherer. 2018. NADiA: Neural Network Driven Virtual Human Conversation Agents. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents (IVA '18)*. ACM, New York, NY, USA, 173–178. https://doi.org/10.1145/3267851.3267860

[44] Vicky Zayats, Mari Ostendorf, and Hannaneh Hajishirzi. 2016. Disfluency Detection using a Bidirectional LSTM. *CoRR* abs/1604.03209 (2016). arXiv:1604.03209 http://arxiv.org/abs/1604.03209

[45] Simon Zwarts, Mark Johnson, and Robert Dale. 2010. Detecting Speech Repairs Incrementally Using a Noisy Channel Approach. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING '10)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 1371–1378. http://dl.acm.org/citation.cfm?id=1873781.1873935

Received November 2018; revised February 2019; accepted April 2019.