CONVO: What does conversational programming need?

Jessica Van Brummelen
jess@csail.mit.edu
MIT

Kevin Weng
kweng@mit.edu
MIT

Phoebe Lin
phoebelin@gsd.harvard.edu
Harvard Graduate School of Design

Catherine Yeo
cyeo@college.harvard.edu
Harvard University

Abstract—**Vast improvements in natural language understanding and speech recognition have paved the way for conversational interaction with computers. While conversational agents have often been used for short goal-oriented dialog, we know little about agents for developing computer programs. To explore the utility of natural language for programming, we conducted a study (n=45) comparing different input methods to a conversational programming system we developed. Participants completed novice and advanced tasks using voice-based, text-based, and voice-or-text-based systems. We found that users appreciated aspects of each system (e.g., voice-input efficiency, text-input precision) and that novice users were more optimistic about programming using voice-input than advanced users. Our results show that future conversational programming tools should be tailored to users' programming experience and allow users to choose their preferred input mode. To reduce cognitive load, future interfaces can incorporate visualizations and possess custom natural language understanding and speech recognition models for programming.**

Index Terms—**conversational programming, conversational AI, interaction paradigms, voice interfaces, accessibility, education, natural language processing, human-computer interaction**

I. INTRODUCTION AND RELATED WORK

With recent major advances in automatic speech recognition (ASR) and natural language processing (NLP) [1]–[3], interacting with technology has become as easy as having a conversation. Agents now automate simple, few-turn tasks, like turning on lights, as well as longer, more complex tasks, such as booking clients' appointments through conversation [4]. Conversational technology can increase accessibility through question-answering (QA), provide alternative input and output methods, and enable interaction without requiring reading/writing skills [5]–[7]. This has positive implications to computer programming as it could lower the barrier to entry (e.g., with no syntax requirements and efficient, alternative input methods) [8]–[10].

However, ASR is still not near the level of recognition humans expect, and can be frustrating for those with non-traditional accents [11]. Additionally, natural language (NL) is innately ambiguous, which could produce additional errors (e.g., is the meaning of "say variable var", *say the value of variable var* or *say the words "variable var"*?) [12], [13].

As a result, little is known about the suitability of conversational AI as a tool for novice programmers. There has been some work in single-turn program synthesis [14], [15]; syntax- or keyword-dependent, voice-based programming [16]–[19]; conversational agents for controlling/developing specific systems, such as robots [8], [20]–[22]; and for learning linear tasks, such as sending emails [23]. Optimal interaction paradigms for NL, multi-turn, conversational systems, on the other hand, remain largely unstudied.

To understand the needs of multi-turn, conversational programming systems, we created a conversational programming tool, CONVO, to investigate the effectiveness of voice- vs. text-based interaction and address primary research questions:

RQ1: What is the preferred input modality for a conversational tool? Would multimodal input be useful?

RQ2: How do input modalities affect cognitive load?

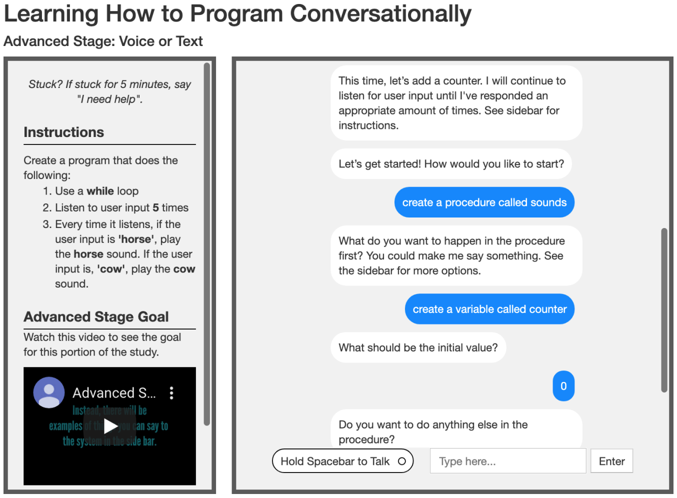
RQ3: How do preferences of novice and advanced programmers for conversational programming differ?

Additionally, we investigated the adequacy of current ASR (i.e., Google Cloud Speech-to-Text API) and constrained NL for conversational coding. This paper explores conversational AI design spaces with respect to lowering the barrier to entry to programming, and presents (1) the system design of a conversational programming agent, (2) a formative study (n=45) examining cognitive load, input modalities, and advanced and novice programmers' performance with a conversational programming agent, CONVO, and (3) design considerations for future systems based on quantitative and thematic analyses.

II. SYSTEM DESIGN

CONVO is a voice-based system allowing users to develop computer programs by conversing in natural language with a conversational agent. For the user study, the system was designed to support both voice- and text-based conversations, which enable three main tasks—program creation, program editing, and system feedback—through natural language. To illustrate, users might say, "Create a variable" and the system would reply, "What do you want to call the variable?", and so on, until the program is complete (e.g., see Fig. 1). The user may then ask to execute the program or to go back and edit. CONVO supports the following *actions* being added to user programs: creating, setting and modifying variables; creating conditionals and loops; saying phrases; playing sounds; and getting user input (i.e., storing a number or phrase stated by the end-user). Other example conversations are shown in [24].

978-1-7281-6901-9/20/$31.00 ©2020 IEEE



Learning How to Program Conversationally

Advanced Stage: Voice or Text

Stuck? If stuck for 5 minutes, say "I need help".

Instructions

Create a program that does the following:

1. Use a **while** loop
2. Listen to user input **5** times
3. Every time it listens, if the user input is **'horse'**, play the **horse** sound. If the user input is, **'cow'**, play the **cow** sound.

Advanced Stage Goal

Watch this video to see the goal for this portion of the study.

Advanced S...
Instead, there will be examples of th you can say to the system in the side bar.

This time, let's add a counter. I will continue to listen for user input until I've responded an appropriate amount of times. See sidebar for instructions.

Let's get started! How would you like to start?

create a procedure called sounds

What do you want to happen in the procedure first? You could make me say something. See the sidebar for more options.

create a variable called counter

What should be the initial value?

0

Do you want to do anything else in the procedure?

Fig. 1. The advanced stage of the study using the voice-or-text-based system, which shows both the record button and text box for input.

CONVO consists of four modules: the voice-user interface (VUI), natural language understanding (NLU) module, dialog manager (DM), and program manager (PM). The VUI receives and transcribes voice input into text using Google's Cloud Speech-To-Text API [25] (if in a voice mode) and has a text box input (if in a text mode). The transcribed text is displayed on screen and sent to the NLU, which is syntactically-constrained and uses a regex expression-based semantic parser to determine intent and extract semantic information. The intent is sent to the DM, which keeps track of the conversation, system- and user-goals, and agent state; sends program-related information to the PM; and generates appropriate responses. Responses are shown on screen and (if in voice mode) voiced back to users using Google's Speech Synthesis API [26]. The PM stores *actions* users specify for their program using a special program representation that can be converted to other formats (e.g., JavaScript, Python). If the user asks to run their program, the PM executes a Python version of the program. Users can scroll through the conversation, as shown in Fig. 1.

III. USER STUDY

We conducted a user study to evaluate the effectiveness of CONVO and to understand the user needs of a conversational programming environment. We recruited 45 participants, who were given $20 or $30 Amazon gift cards depending on whether they self-identified as novice or advanced (as the length of the study increased if advanced).

Participants interacted with CONVO in three stages: the practice stage (create a program where CONVO says "hello world"), novice stage (create a program where CONVO listens for user input and plays two different animal sounds—e.g. If I say "cat", play "meow") and advanced stage (create a program where CONVO continuously listens for user input a set number of times and plays corresponding animal sounds) only for advanced users. At each stage, participants engaged in video and in-context tutorials, had access to view documentation in a sidebar, and interacted with the three input systems to complete programming goals. Participants could not move onto the next goal until CONVO programmatically checked for current-goal completion.

We performed a mixed between- and within-subject test, where the between-subject conditions were between novice and advanced participants and the within-subject condition was input modality type. We introduced slight goal variations and randomized the order of the systems (voice-input, text-input and voice-or-text systems) to account for learning effects.

Each participant completed a questionnaire about their demographic and programming background. We recorded participants' time to goal completion, text- and voice-input transcripts, number of times they asked for help, and the number of resets of the goal (e.g., when they got stuck). We also recorded 5-point Likert-scale preferences about usability, satisfaction, and efficiency (as shown in Fig 4) and comparisons of the three systems, as well as responses to free-form questions about likes/dislikes, features they wished to add, questions they had about CONVO, and challenges they ran into.

We performed quantitative and qualitative analyses of the data, using analysis of variance (ANOVA) for between-subjects analyses, repeated measures ANOVA for within-subjects analyses[1], and an inductive approach [27] (open coding) for free-form responses. We identified fourteen design themes[2], with the most common shown in Fig. 2 and 3.

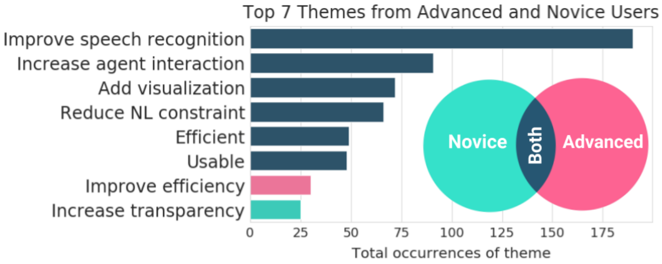


Fig. 2. Total number of occurrences for the top seven themes from advanced user responses and top seven from novice user responses. Novice responses emphasized transparency over efficiency. Note how the colors represent which user group(s) the theme came from (e.g., pink represents a top theme from novice users, dark blue is from both novice and advanced users).

IV. RESULTS AND DESIGN RECOMMENDATIONS

Through the quantitative and qualitative analyses, we identified six main design recommendations for future conversational programming systems.

Tailor to programming experience and task (RQ 3). Our results suggest that conversational programming systems should be tailored to their audiences due to differences in user preferences and abilities. We found that novice users generally found voice-input useful and enjoyable, whereas advanced users tended to view it less so, as shown in Fig. 4 and 5. Furthermore, although there was no significant difference

[1] See appendix [24] for more detail about the quantitative analyses.

[2] See appendix [24] for details about the design themes.

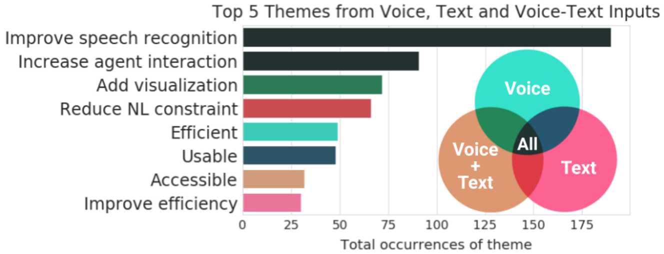


Fig. 3. Total number of occurrences for the top five themes from each system survey. The voice-input system responses emphasized efficiency; text-input, a need to improve efficiency; and voice-or-text, accessibility. Note how the bars' colors represent which input system(s) the theme came from (e.g., pink represents text-input system, and green represents voice-input and voice-or-text systems).

between the overall number of voice- and text-inputs, in the advanced stage users tended to type rather than speak (p=0.003). Advanced users also perceived voice-or-text to be more difficult than text (p=0.02), but there was no significant difference for novice users. Thus, for an advanced audience, it may be more important to have a text-input option than for a novice audience, and for an introductory audience, a voice-input system may be more useful than for an advanced one.

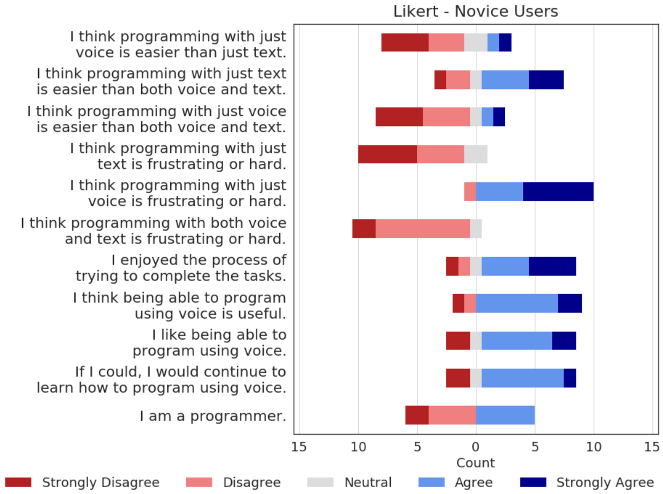


Fig. 4. Novice responses to Likert scale questions. Novices generally found voice useful and enjoyable. Refer to Section V for further discussion.

We also found that some advanced users found NL programming cumbersome, likely because they were used to syntax-restricted programming languages (e.g., "It also seems quite inefficient to figure out the right way to express a statement in actual words that otherwise can be typed in a programming language using very specialized characters."), whereas novice users tended to praise the naturalness of the language (e.g., "I liked the simplicity of using the normal talk, as in not coding necessarily"). This is further reflected in how "improve efficiency" was found in advanced users' top seven themes, but not novice users' (see Fig. 2). Thus, NL may be a better fit for an educational, introductory tool than an advanced tool.

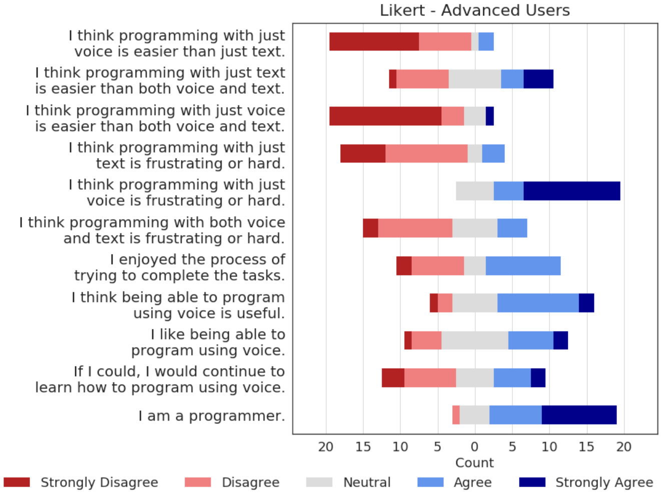


Fig. 5. Advanced user responses to Likert scale questions. Advanced user responses tended to be less favorable towards voice than novice responses.

Design a flexible, accessible system (RQ 1). Our results suggest conversational programming systems should be accessible through both voice- and text-input. Participants found value in both modalities, often citing voice as efficient (see Fig. 3) and text as accurate. Many participants had comments similar to, "I liked being able to use the voice for longer commands, and the text for shorter commands or misunderstood commands". This was supported by the significant difference in number of characters (p=0.004) and words (p=0.003) per voice utterance over text utterance (i.e., longer voice utterances). Furthermore, when using the voice-or-text system, participants used both voice and text input, and there was no statistical evidence for a difference in how many times participants spoke versus typed.

From an accessibility standpoint, it makes sense to provide both input options, and allow each of them to stand alone (such that the system is completely accessible by voice-only and text-only). With current technologies, however, this may be difficult to achieve. The Google Cloud Speech-to-Text [25] ASR system we used—which is often recognized as the gold standard [28], [29]—did not seem sufficient for programming. Many participants commented on this (e.g., "Sometimes it had problems understanding my speech, so I resorted to typing things.", "It seems like if speech recognition worked well, it would be a better choice, but having this [text option] is useful") and we found that the most common theme was to *improve speech recognition*. Thus, until speech recognition systems improve, it may be infeasible to have a standalone voice-input system.

Design a transparent system (RQ 3). Many participants described how they would appreciate being able to ask the system how it works. Some questions included:

- "What kind of nueral [*sic*] network do you run on?"
- "How do you understand what I'm saying?"
- "How do you map my phrases to commands?"
- "What kind of voice recognition is used?"

- "Why didn't the agent understand me?"
- "Do you use any sort o [*sic*] machine learning to recognize the accents?"

Transparency was one of the top occurring themes for novice users (see Fig. 2) and is especially important when developing AI systems for novice programmers and education.

Design with visualizations (RQ 1). A common theme in the free-form responses was the desire for code visualizations. This was in the top seven commonly occurring themes for both novice and advanced users, and the top five themes for both the voice- and text-based systems. Specifically, users asked for ways to "visualize where [they] are in the program", view a "representation of the code [they were] making", "see [...] variable names or the name of the procedure", "see which level [they]'re at [in the program]", and visually "modify [their] previous lines that were misinterpreted". As current technology focuses heavily on visual systems and computer screens, voice-only systems can force high memorization requirements on users. Nonetheless, depending on the intended audience, one may choose to avoid visualizations or make them non-essential to the system for accessibility reasons.

Design to reduce cognitive load (RQ 2). In the thematic analysis, some participants mentioned high cognitive load due to a lack of visualizations (e.g., "I found it quite challenging to figure out the logic of the program entirely in my head; [...] it felt like I had to figure it all out before entering anything."). In future studies, we will analyze cognitive load effects of integrating visualizations into CONVO. We expect this will reduce the cognitive load for sighted users. Other design features to potentially reduce cognitive load include decreasing the constraint on the NL input such that users will no longer have to remember specific phrases, and improving the speech recognition model such that people don't have to repeat phrases as often, and are more likely to remember where they are in the program.

For all cognitive load indicators (number of resets of the system, time to goal completion, and number of times users asked for help), we found no evidence for a significant difference between the voice-based, text-based, and voice-or-text-based systems; thus, each system may be a viable option when designing for cognitive load.

Improve ASR and NLU (RQ 1). The most common theme in the free-form responses was to improve speech recognition. As mentioned previously, we used the Google Cloud Speech-to-Text [25] ASR system—which is often recognized as the top online ASR [28], [29]—for CONVO. Evidently, current ASR systems are not sufficient for fully standalone voice-based, NL programming systems. One potential avenue for improvement is to develop a custom NL programming ASR model that incorporates common NL programming phrases, like "create a variable", to ensure recognition of those phrases. Nonetheless, by training on specific phrases, this may cause the model to be less robust to new phrases, which would somewhat defeat the purpose of a generalizable NL system.

In addition to improved speech recognition, participants desired reduced constraint on NL input (e.g., "It's a very cool idea, and with expanding the dictionary it could work better.", "I expect more natural-language input support such as 'nope', 'no thanks', etc. would be valuable as well."). Reducing NL constraint was a top theme in both the text-based and voice-or-text-based systems, as well as in both novice and advanced users' responses (see Fig. 2 and 3).

We are currently developing an unconstrained NL version of CONVO to understand whether this improves or reduces performance, as there has been research questioning the suitability of unconstrained NL for programming [12], [13]. Nevertheless, with additional ambiguity reduction techniques, such as conversational QA and immediate feedback from the agent, unconstrained NL may become suitable for introductory, educational NL programming, especially due to the positive feedback in this area from the free-form responses (e.g., "It gives feedback, which is really useful", "The process is pretty interactive and fun. The idea of using natural language to code is great and the system reacts very fast.", "Feedback is immediate.").

V. LIMITATIONS

Although this study aims to address RQs 1-3 with respect to conversational programming in general, aspects of CONVO inevitably influenced the results. Thus, we encourage further studies with variations on conversational programming systems. Additionally, frequent prior experience with text-based systems and programming languages may have led to a strong bias in favor of the text-based system, especially for advanced users.

Additionally, there was a seeming contradiction with the Likert scale results due to negative responses to "voice being easier than text or voice-or-text", but positive responses to "voice being useful", "liking being able to use voice", and "wanting to continue learning how to program using voice", especially for novice users. Nonetheless, according to the free-form responses, this could be explained by participants' frustration with ASR, but commendation of the efficiency of the voice system (e.g., "When [voice recognition] worked, it seemed it could be faster than typing", "It seems like if speech recognition worked well, it would be a better choice",). This explanation is further supported by both "improve speech recognition" and "efficient" being in the top five most common voice-based system themes.

VI. CONCLUSIONS

In this study, we investigated the effectiveness of voice-based, text-based, and voice-or-text-based systems in a conversational programming environment in terms of difficulty, efficiency, and cognitive load indicators through free-form responses, Likert scale questions, and user activity. Our results show a desire for and optimism about conversational programming, especially in introductory programming systems.

VII. ACKNOWLEDGEMENTS

We would like to thank Hal Abelson, Marisol Diaz, Selim Tezel, and Ilaria Liccardi for their support, as well as the participants in our study for their time.

References

[1] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-based models for speech recognition,” in *Advances in neural information processing systems*, pp. 577–585, 2015.

[2] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, *et al.*, “Deep speech 2: End-to-end speech recognition in english and mandarin,” in *International conference on machine learning*, pp. 173–182, 2016.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *NAACL-HLT*, 2019.

[4] D. E. O’Leary, “Google’s duplex: Pretending to be human,” *Intelligent Systems in Accounting, Finance and Management*, vol. 26, no. 1, pp. 46–53, 2019.

[5] T. J.-J. Li, “End user programing of intelligent agents using demonstrations and natural language instructions,” in *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion*, pp. 143–144, 2019.

[6] A. Stefik, C. Hundhausen, and R. Patterson, “An empirical investigation into the design of auditory cues to enhance computer program comprehension,” *International Journal of Human-Computer Studies*, vol. 69, no. 12, pp. 820 – 838, 2011.

[7] A. Armaly, P. Rodeghero, and C. McMillan, “A comparison of program comprehension strategies by blind and sighted programmers,” *IEEE Transactions on Software Engineering*, vol. 44, no. 8, pp. 712–724, 2017.

[8] H. Jung, H. J. Kim, S. So, J. Kim, and C. Oh, “Turtletalk: an educational programming game for children with voice user interface,” in *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–6, 2019.

[9] K. T. H. Rahit, R. H. Nabil, and M. H. Huq, “Machine translation from natural language to code using long-short term memory,” in *Proceedings of the Future Technologies Conference*, pp. 56–63, Springer, 2019.

[10] A. Desilets, “Voicegrip: a tool for programming-by-voice,” *International Journal of Speech Technology*, vol. 4, no. 2, pp. 103–116, 2001.

[11] S. Mccrocklin, “Learners’ feedback regarding asr-based dictation practice for pronunciation learning,” *CALICO Journal*, vol. 36, no. 2, pp. 119–137, 2019. Copyright - Copyright Equinox Publishing Ltd. 2019; Last updated - 2019-09-03.

[12] M. G. Helander, *Handbook of human-computer interaction*. Elsevier, 2014.

[13] J. Good and K. Howland, “Programming language, natural language? supporting the diverse computational activities of novice programmers,” *Journal of Visual Languages & Computing*, vol. 39, pp. 78–92, 2017.

[14] M. Rabinovich, M. Stern, and D. Klein, “Abstract syntax networks for code generation and semantic parsing,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1139–1149, 2017.

[15] E. C. Shin, M. Allamanis, M. Brockschmidt, and A. Polozov, “Program synthesis and semantic parsing with learned code idioms,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 10825–10835, Curran Associates, Inc., 2019.

[16] A. Begel and S. L. Graham, “An assessment of a speech-based programming environment,” in *Visual Languages and Human-Centric Computing (VL/HCC’06)*, pp. 116–120, IEEE, 2006.

[17] A. Nowogrodzki, “Writing code out loud,” *Nature*, vol. 559, pp. 141–142, Jul 05 2018.

[18] R. Hileman, “Talon.” https://talonvoice.com/, 2018. Accessed: 2020-05-21.

[19] T. Rudd, “Using python to code by voice.” https://youtu.be/8SkdfdXWYaI, 2013. Accessed: 2020-05-21.

[20] Y. Kim, Y. Choi, D. Kang, M. Lee, T.-J. Nam, and A. Bianchi, “Heyteddy: Conversational test-driven development for physical computing,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 4, pp. 1–21, 2019.

[21] J. Thomason, A. Padmakumar, J. Sinapov, N. Walker, Y. Jiang, H. Yedidsion, J. Hart, P. Stone, and R. J. Mooney, “Jointly improving parsing and perception for natural language commands through human-robot dialog,” *Journal of Artificial Intelligence Research*, vol. 67, pp. 1–48, 2020.

[22] S. Pérez-Soler, E. Guerra, and J. de Lara, “Collaborative modeling and group decision making using chatbots in social networks,” *IEEE Software*, vol. 35, no. 6, pp. 48–54, 2018.

[23] T. J.-J. Li, I. Labutov, B. A. Myers, A. Azaria, A. I. Rudnicky, and T. M. Mitchell, “An end user development approach for failure handling in goal-oriented conversational agents,” *Studies in Conversational UX Design*, 2018.

[24] J. Van Brummelen, “Appendix for Convo: What does conversational programming mean?,” 2020. https://gist.github.com/jessvb/30271c1b7885a517296d1d30c566ea3f, Last accessed on 2020-05-20.

[25] Google, “Google cloud speech-to-text,” 2020. https://cloud.google.com/speech-to-text, Last accessed on 2020-02-24.

[26] Google, “Introduction to the speech synthesis api,” 2014. https://developers.google.com/web/updates/2014/01/Web-apps-that-talk-Introduction-to-the-Speech-Synthesis-API, Last accessed on 2020-02-24.

[27] D. R. Thomas, “A general inductive approach for analyzing qualitative evaluation data,” *American Journal of Evaluation*, vol. 27, no. 2, pp. 237–246, 2006.

[28] J. Y. Kim, C. Liu, R. A. Calvo, K. McCabe, S. C. Taylor, B. W. Schuller, and K. Wu, “A comparison of online automatic speech recognition systems and the nonverbal responses to unintelligible speech,” *arXiv preprint arXiv:1904.12403*, 2019.

[29] V. Këpuska and G. Bohouta, “Comparing speech recognition systems (microsoft api, google api and cmu sphinx),” *Int. J. Eng. Res. Appl*, vol. 7, no. 03, pp. 20–24, 2017.