

Conversational Agents to Democratize Artificial Intelligence

Jessica Van Brummelen

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
jess@csail.mit.edu

I. INTRODUCTION

Artificial Intelligence (AI) technology can be found nearly everywhere. However, developing and controlling this technology is generally limited to large companies or those with extensive training in computer science. It is difficult for many who have application ideas for AI to even know where to start when developing AI technology.

In [7], I empower students with little to no programming experience to develop conversational AI agents (AI programs that communicate with humans using natural language). Through high school workshops and block-based coding tools, students created agents to help people remember forgotten words, determine how to recycle items, and learn math concepts. Students learned computational thinking (CT) and machine learning (ML) concepts through developing these projects.

Nevertheless, this system has limitations. It uses the Alexa Skills platform, which requires users to have access to Alexa Developer and Amazon Web Services accounts. As well, to speak to the agents, one must begin all conversations with the word, “Alexa”, which can be frustrating. Furthermore, the visual coding tool limits its user-base to those with regular vision and those who know how to use a traditional computer.

Moving forward, I plan to develop a voice-based, conversational platform to empower an even wider audience — including those with vision impairments and those without traditional computing experience — to develop computation/data structures, such as loops, conditionals, and objects, as well as ML models, through natural interaction. I will explore the effectiveness of a voice-based system compared to current visual- and text-based coding tools to provide a low barrier to entry to programming, teach CT skills, and engage those with little to no experience coding in STEM-related activities.

II. VOICE-BASED AND NATURALISTIC PROGRAMMING

The majority of voice-based programming tools are for advanced programmers, require knowledge of programming languages, and/or are not in natural language. For instance, the voice-based programming systems, *VocalIDE*, *VoiceCode*, and *Talon*, all require an understanding of computer programming concepts, such as programming syntax or specific language keywords [4], [6]. Furthermore, the systems were designed

for people with normal vision, and do not include voiced feedback, but rather require visual inspection of code.

Current naturalistic programming tools (which use human language, like English, rather than a specific programming language) also frequently lack voiced feedback. These systems, such as *Metafor*, *Pegasus*, and *Macho*, tend to be command-line or GUI-based programs. One promising aspect of *Metafor*, however, is that it converses (via text) with the user to resolve ambiguity. Nonetheless, *Metafor*, *Pegasus*, and *Macho* were developed for a knowledgeable programmer audience [5].

Other voice-based systems have been developed for a wider audience. For instance, visually-impaired and sighted students co-designed voice-based systems for education in [3]. Although these systems were designed for pedagogy, they were not designed to teach programming skills. A different voice-based system for teaching such skills was developed in [8]; however, it was heavily dependent on visual feedback.

I propose developing a conversational, voice-based, naturalistic language system to empower nearly anyone — even those with vision impairments — to program applications through conversational AI.

III. CONVERSATIONAL AI IN MIT APP INVENTOR

In [7], I developed a programming tool for a non-programmer audience to develop conversational agents. This tool utilizes the block-based, visual coding interface of MIT App Inventor to lower the barrier to entry to programming, teach CT skills, and empower students to create their own conversational agent applications. Additionally, within these applications, students can use ML (through pretrained, long short-term memory networks) to generate responses. Similarly, the voice-based, naturalistic programming environment will provide a low barrier to entry by removing syntax errors, democratize AI through providing access to ML networks, and teach students CT skills as they develop projects. Through these tools, I will compare how students learn CT skills through voice-based versus visual programming.

IV. COMPUTATIONAL THINKING & CONVERSATIONAL AI

CT frameworks are generally used to develop computer science curricula and outline skills necessary for students to become successful programmers. Although these frameworks are generally derived to teach people about programming, they may also be conceptualized as outlines for the necessary

components of a programming language. To illustrate, the framework in [1] describes seven CT concepts, *sequences*, *loops*, *parallelism*, *events*, *conditionals*, *operators*, and *data*, which are all fundamental to a mature programming language. Dialogs 1, 2, and 3 illustrate how some of these concepts, including sequences, loops, and events, may be implemented in a conversational, naturalistic programming environment.

Dialog 1. Example routine creation

A¹: "Hello! What would you like to create or do today?"
 U: "I want you to get my room ready for playtime."
 A: "How do I do that?"
 U: "First, you turn on the lights."
 A: "Got it. What's next?"
 U: "Then play my favorite song."
 A: "What's your favorite song?"
 U: "Count on Me"
 A: "By Bruno Mars?"
 U: "Yep."
 A: "Okay, I saved your favorite song to be Count on Me. After playing that, what should I do?"
 U: "Make my robot dance while you play the song."
 A: "Okay, I'll do your make robot dance routine while your favorite song plays. What's next?"
 U: "That's it!"
 A: "Okay, I will start the get room ready for playtime routine."

Dialog 2. Example object creation

A: "Hi there! What would you like to make or do today?"
 U: "I want to make a story!"
 A: "What's a story? Can you describe it in terms of my fundamental data types or other things you've created?"
 U: "What are fundamental data types?"
 A: "My fundamental data types are numbers, letters, sentences, lists, and true-or-falses."
 U: "A story is a bunch of sentences."
 A: "Okay, so a story is a list of sentences, correct?"
 U: "Yes."
 A: "Alright, I will remember what a story is. Do you want to make a specific story now?"
 U: "Yes."
 A: "What do you want the first sentence to be?"
 U: "Once upon a time, a little girl found a small plant."
 A: "Okay, what's the next sentence?"
 ...
 A: "To save the story, I need to know its name. What do you want to call your story?"
 U: "Sweetgum Tree Adventures."

Dialog 3. Example event creation

U: "When it's bedtime, read me Sweetgum Tree Adventures."
 A: "How do I know it's bedtime?"
 U: "Bedtime is at 8 pm."
 A: "Okay, I will read, Sweetgum Tree Adventures when it's bedtime."

V. FUTURE WORK

A common problem in naturalistic programming is dealing with ambiguity. As illustrated in Dialog 1, when the agent encounters an ambiguous or unknown noun or verb, it asks the user for clarification. This technique is similar to *slot filling*. In slot filling, agents search phrases for potential candidates for slot types (for example, the word, "Canada", as a candidate for the slot type, "Country"). Since the agent will not necessarily be searching for a specific slot type, an efficient method to parse and search for unknown/ambiguous words will need to be implemented.

¹A stands for agent and U stands for user.

Furthermore, I will implement a natural language understanding (NLU) system. This may involve ML-based methods, symbolic methods, or combinations of both. For example, although transformer networks have been shown to be effective at syntactic parsing [2], I will need to ensure specific concepts, such as phrases that indicate conditionals, like *if* and *since*, are parsed and understood correctly. The seven CT concepts from [1] will need special attention, which may involve developing a set of rules to complement a standard NLU system.

Other work will include ensuring the agent adheres to conversational AI principles. In [7], I define these as the *flexible*, *concise*, *correct*, *relevant*, *natural*, *social extent*, *direct definition*, and *ethical* principles. Specific attention to the *concise* and *correct* principles will be taken. Together, these two principles articulate how conversational agents should share an appropriate amount of information based on the current context. For instance, a new user may need an explanation of the agent's *fundamental data types* (as shown in Dialog 2), whereas an experienced user might not.

The effectiveness of the proposed voice-based system will be tested through Wizard of Oz experiments, general user testing, and field studies. The main aspects to be tested include:

- Barrier to entry compared to block- and text-based coding
- Non-expert programmer experience of the system
- Experience of those with visual impairments
- Ability of the system to teach CT skills
- Ability of the system to teach ML concepts

VI. CONCLUSIONS

Through a conversational, naturalistic programming tool, I plan to investigate the effectiveness of voice to reduce the barrier of entry to programming, democratize AI, and teach CT skills.

ACKNOWLEDGMENT

This work was supported by an Alexa Graduate Fellowship.

REFERENCES

- [1] Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*, volume 1, page 25, 2012.
- [2] Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. *ACL*, 2018.
- [3] Oussama Metatla, Alison Oldfield, Taimur Ahmed, Antonis Vafeas, and Sunny Miglani. Voice user interfaces in schools: Co-designing for inclusion with visually-impaired and sighted pupils. In *Proceedings of CHI on Human Factors in Computing Systems*. ACM, 2019.
- [4] Anna Nowogrodzki. Speaking in code: how to program by voice. *Nature*, 559(7712):141, 2018.
- [5] Oscar Pulido-Prieto and Ulises Juárez-Martínez. A survey of naturalistic programming technologies. *ACM Computing Surveys*, 50(5):70, 2017.
- [6] Lucas Rosenblatt. VocalIDE: An IDE for programming via speech recognition. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 417–418. ACM, 2017.
- [7] Jessica Van Brummelen. Tools to create and democratize conversational artificial intelligence. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2019.
- [8] Amber Wagner and Jeff Gray. An empirical evaluation of a vocal user interface for programming by voice. *International Journal of Information Technologies and Systems Approach (IJITSA)*, 8(2):47–63, 2015.