Virtual Summer Camp for High School Students with Disabilities – An Experience Report

Wei Wang
The University of Texas at San Antonio
USA
wei.wang@utsa.edu

Kathy B. Ewoldt
The University of Texas at San Antonio
USA
kathy.ewoldt@utsa.edu

Mimi Xie
The University of Texas at San Antonio
USA
mimi.xie@utsa.edu

Alberto M. Mestas-Nuñez
The University of Texas at San Antonio
USA
alberto.mestas@utsa.edu

Sean Soderman
The University of Texas at San Antonio
USA
sean.soderman@my.utsa.edu

Jeffrey Wang
Keystone School
USA
jeffreywang51@gmail.com

ABSTRACT

In the past years, the authors held computer programming and machine learning summer camp for high-school students with disabilities. Due to the pandemic, the summer camp was offered virtually in 2020 and 2021. This paper reports our experience of teaching this summer camp. The main goal of the summer camp was to let students with disabilities get first-hand experience of working in STEM fields to encourage them to pursue STEM careers. The curriculum was primarily composed of hands-on activities for Python programming and Computer Vision. Besides lectures and programming tasks, there were also guest speakers and an external panel to offer their personal experiences of working in STEM fields.

Students enrolled had disabilities including Autism, learning disability, vision impairments, and blindness. The summer camp was successful, and all students were able to finish all programming tasks. Special accommodations and adjustments were necessary, especially in the curriculum, schedule, teaching practices, and the involvement of special educators. The overall effort to make these adjustments was not extremely high. However, we did observe accessibility issues for the student with blindness. Overall, our experience shows that online programming camps for high-school students with disabilities are feasible, and students with disabilities can potentially be successful Computer Science professionals.

CCS CONCEPTS

• **Applied computing → Distance learning**; • **Social and professional topics → K-12 education**; **Model curricula**.

KEYWORDS

Special Education, K-12 Education, Programming Education, Machine Learning Education

ACM Reference Format:
Wei Wang, Kathy B. Ewoldt, Mimi Xie, Alberto M. Mestas-Nuñez, Sean Soderman, and Jeffrey Wang. 2023. Virtual Summer Camp for High School Students with Disabilities – An Experience Report. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023), March 15–18, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3545945.3569818

1 INTRODUCTION

Motivation Computer Science (CS) occupations, such as software engineers and data scientists, are among the highest-paid jobs in the U.S. Moreover, many of these jobs usually only require a desk and a computer, and thus are flexible with work modality. Even before the pandemic, many software engineers and data scientists were already working remotely from home [9, 46, 54]. Because of the high pay and flexibility, Computer Science careers can be perfectly viable for people with disabilities. Moreover, if more people with disabilities can be directly involved in Computer Science and software development, it may also significantly improve the usability and accessibility of computer software.

To encourage high-school students with disabilities to pursue Computer Science and other STEM careers, we had offered a computer programming and machine learning summer camp, called *ExploreSTEM*, to them. Due to the pandemic, the summer camp was offered virtually (online) in 2020 and 2021. In this paper, we report our experience and lessons learned from teaching *ExploreSTEM* virtually, with the hope that this report can help other educators when planning similar virtual summer camps.

Summary of the Summer Camp. Our summer camp provided lectures and hands-on activities for Python programming and Computer Vision to let the students experience CS careers as software engineers and data scientists. The hands-on activities included seven programming tasks covering Python programming, image segmentation, hand-written digit recognition, and lane and vehicle detection for autonomous driving. Google Colaboratory (Colab) was used as the programming platform, with lectures and talks delivered via Microsoft Teams or Zoom synchronously. Besides the programming activities, the summer camp also included guest presentations and a panel, where the guests and panelists who worked in Computer Science and other STEM fields presented their personal career experiences and career paths.

The camp in 2020 was half-day and lasted for two weeks with nine students enrolled. The camp in 2021 was full-day and lasted for one week with six students. All students were 14+ years old high school students with disabilities including Autism, learning disabilities, and blindness and vision impairments (BVI). The instruction team of *ExploreSTEM* included faculty members with expertise in Computer Science, Special Education, and Earth Science, as well as several undergraduate and graduate student tutors. There was also a high school student who evaluated the difficulty of the programming activities before the camp started.

Outcomes The summer camp in both years was successful. All students were able to finish all programming tasks. Our students also expressed positive feelings regarding the camp. The outcomes are summarized in the following list,

(1) All students correctly finished all programming tasks, although some students did need additional time and assistance in debugging. The majority of the students also showed good understanding regarding programming and machine-learning/AI concepts.
(2) Students also expressed that they enjoyed the career experience of Computer Science and the summer camp. One student returned for the second year's camp.
(3) Overall, our experience shows that online programming camps for high-school students with disabilities are feasible, and students with disabilities can potentially be successful Computer Science professionals. We also feel the effort required to adjust the online teaching for students with disabilities is not extremely high.

Lessons Learned The lessons-learned regarding curriculum, programming activities, and schedule include,

(1) The programming activities should reduce the length of the statements, the number of punctuation marks, and the use of identifiers that mix letters, digits, and underscores. Each block of code should include no more than 5 lines of statements and should be independently executable/verifiable to simplify debugging. These simplified programming activities are particularly important for students with Autism and blindness, as they avoided over-stressing the students and were more friendly to screen readers.
(2) Extra tutoring time can help alleviate issues from students' different programming progresses. The differences in the teaching needs among the students lead to differences in progress. Therefore, we found that individual tutoring sessions were particularly helpful for students who needed more time to code without impeding the teaching for others.
(3) Guest speaker presentations and the panel discussion were valuable additions to the lectures and hands-on activities. They provided the students with personal career experience in other STEM areas and career-building advice. More importantly, they provided an opportunity for the students to discuss the difficulties beyond academia, such as working environments, accommodations, and financial support.
(4) To encourage students with disabilities to attend colleges, discussion of potential financial support is also important. One of the major topics brought up during our panel discussion was financial support, which turned out to be a major obstacle for students with disabilities in attending college. Our panelists mentioned several supports they have used, which were less-known to our students, their parents, and even some social workers.

The lessons learned for technology/accessibility include,

(1) Google Colab, or more generally, cloud-based Jupyter Notebooks, greatly reduced the difficulty of teaching and debugging assistance, which is also consistent with our own experience of teaching programming in courses for typical students and with other reports [2].
(2) The student with blindness required a screen reader to read code examples, as well as his own code. However, the screen reader had trouble providing comprehensible readings for program statements that mixed letters, digits, and punctuation marks. Moreover, the student also had trouble navigating inside his own code. During teaching, our instructors and tutors frequently had to read the example codes, instead of using the screen reader. We also needed to monitor and assist his cursor movement in the code. These difficulties showed that there needed better technological support for screen reading and code navigation to assist students with BVI to learn Computer Science, which is also consistent with other reports [5, 6, 15, 16, 39, 50].
(3) Image outputs from the code (including images generated by students' code) should have image descriptions, which can be read by screen readers to assist students with blindness.

The presence of a special educator is also indispensable,

(1) Our special education professor provided training for the instructors and tutors, which ensured the camp's success. The training included characteristics of disabilities, accommodations for learning, and strategies that promote engagement. It also prepared them to follow the evidence-based practices for teaching students with disabilities [17, 36, 42].
(2) Despite the training, there still must be a special educator present at the camp to closely monitor the students. As our camp is open to students with any disability, we need to satisfy a variety of teaching needs. Moreover, even with training, CS instructors may still overlook certain requirements. Therefore, our special education professor constantly monitored the teaching and would intervene when necessary, to avoid student over-stressing, to clarify instructions, and to maintain ordered communication.

The rest of this paper is organized as follows: Section 2 gives an overview of the activities of our summer camp; Section 3 summarizes teaching outcomes; Section 4 presents lessons learned; Section 5 discusses related work; and Section 6 concludes the paper.

2 CAMP SYLLABUS

2.1 Camp Schedule

Table 1 and Table 2 give the detailed schedule of our summer camps. As Table 1 shows, the camp in 2020 lasted for two weeks (10 days),

	Hours (AM)	Day 1	Day 2	Day 3	Day 4	Day 5
Week 1	9-11	Prog.	Extra Tutor	Prog.	Extra Tutor	Prog.
	11-12	Guest		Guest		Guest
Week 2	9-11	Prog.	Extra Tutor	Instr.	Extra Tutor	Prog.
	11-12	Guest		Guest		Guest

Table 1: Schedule of the 2020 camp. "Prog" stands for programming activities, and "Guest" stands for guest speakers.

	Hours	Day 1	Day 2	Day 3	Day 4	Day 5
Morning	9-11	Prog.	Prog.	Prog.	Prog.	Prog.
	11-12	Guest	Guest	Guest	Guest	Panel
Afternoon	13-15	Extra Tutor	Extra Tutor	Extra Tutor	Extra Tutor	Extra Tutor

Table 2: Schedule of the 2021 camp. "Prog" stands for programming activities, and "Guest" stands for guest speakers.

with new instructions and programming activities given in the morning every other day. The days (mornings) between two instructional days were reserved as additional tutoring time for students who needed more time to code. For each instructional morning, there was a 2-hour teaching session and a 1-hour guest speaker presentation session. Moreover, each 2-hour session of instruction/programming activities were partitioned into two 45 to 50 minutes blocks with 15-minute breaks in between.

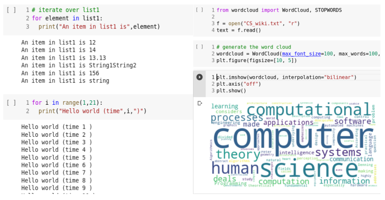
Table 2 shows that the 2021 camp lasted for one week (5 days), with instructions and programming activities in the morning and tutoring in the afternoon. If no one required additional tutoring on a day, the tutoring time was used for backup programming activities.

2.2 Programming Activities

The camp was originally planned as in-person, where the student would build small self-driving robots. However, due to COVID19, the camp has to be converted to online. Therefore, we removed the robot component and chose to emphasize on programming activities and machine learning. The most challenging parts of this change are the debugging support and programming activity complexity. This section presents the programming activities first. The debugging support and programming activity complexity will be discussed in Section 4.

Seven programming activities were provided for Python programming, image segmentation, handwritten-digit recognition, and autonomous driving. Short lectures on the concepts were also given along with the programming activities. For programming activities, the students were first asked to copy (by typing) programs following the example code. After the examples, they were also asked to do exercises independently to further understand the code.

Python Programming – 2 Activities. As Python is the common language for Machine Learning, we started with basic Python programming. Figure 1 shows parts of our Python programming activities. In the first Python activity (Figure 1a), the students learned and programmed basic Python statements, such as variable assignments, math operations, function invocations, if-else statements, and for loops. In the end, the students were asked to modify their loop code and to generate a word cloud with their own paragraphs of text. In the second (and more difficult) Python programming activity, the students were asked to write a "Turtle Racing" drawing



```
# iterate over list1
for element in list1:
  print("An item in list1 is",element)
```

```
An item in list1 is 12
An item in list1 is 14
An item in list1 is 13.13
An item in list1 is String1String2
An item in list1 is 156
An item in list1 is string
```

```
for i in range(1,21):
  print("Hello world (time",i,")")
```

```
Hello world (time 1 )
Hello world (time 2 )
Hello world (time 3 )
Hello world (time 4 )
Hello world (time 5 )
Hello world (time 6 )
Hello world (time 7 )
Hello world (time 8 )
Hello world (time 9 )
```

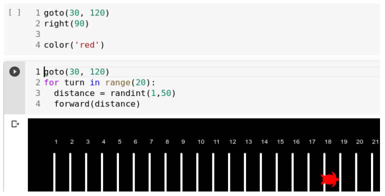
```
from wordcloud import WordCloud, STOPWORDS

f = open("CS_wiki.txt", "r")
text = f.read()
```

```
# generate the word cloud
wordcloud = WordCloud(max_font_size=100, max_words=100,
plt.figure(figsize=[10, 5])
```

```
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

(a) Basic python programming activity.



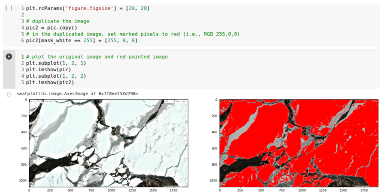
```
goto(30, 120)
right(90)

color('red')
```

```
goto(30, 120)
for turn in range(20):
  distance = randint(1,50)
  forward(distance)
```

(b) Turtle race programming activity.

Figure 1: Python programming activities.



```
plt.rcParams['figure.figsize'] = [20, 20]

# duplicate the image
pic2 = pic.copy()
# in the duplicated image, set marked pixels to red (i.e., RGB 255,0,0)
pic2[mask_white == 255] = [255, 0, 0]
```

```
# plot the original image and red-painted image
plt.subplot(1, 2, 1)
plt.imshow(pic)
plt.subplot(1, 2, 2)
plt.imshow(pic2)
```

```
<matplotlib.image.AxesImage at 0x7f8ee153d290>
```

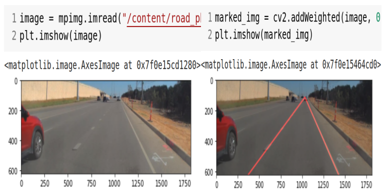
Figure 2: Satellite Image segmentation activity. Left: the original image; right: image with old sea ice identified (in red).

program (Figure 1b) that mimics the Logo language in Python. In the end, the students were asked to increase how far the turtle can go in the field as a competition.

Color-Based Image Segmentation – 2 Activities. As a simple introduction to Computer Vision, the students were tasked to conduct color-based image segmentation. The first segmentation task was to segment the clownfish in a picture. The students were first shown the code of segmenting the orange pixels from the original picture, then they were tasked to isolate white pixels from the picture by themselves. Finally, the segmented orange and white pixels were combined to identify the clownfish. To showcase the real-life use of color-based segmentation, the students also segmented polar sea-ice images obtained with the Sentinel-2 satellite [10] (Figure 2). By isolating white-ish pixels, the students identified old (thick) sea ice from young (thin) sea ice and open water. This activity is derived from our ongoing research on polar sea ice retreat monitoring.

Autonomous Driving – 2 Activities. To illustrate more advanced Computer Vision, the students were tasked to conduct lane detection and vehicle detection for autonomous driving. Both detection tasks used the open data set from Udacity [53]. Example codes for both activities are given in Figure 3. The lane detection activity employed traditional Computer Vision techniques of Gaussian Blur and Hough Transform. The vehicle detection required training a Support Vector Machine (SVM) model. After finishing the detection activities following the example code, the students were asked to write code to randomly pick images to test.

Handwritten Digit Recognition – 1 Activity. To finally showcase working in deep-learning careers, we taught the students the

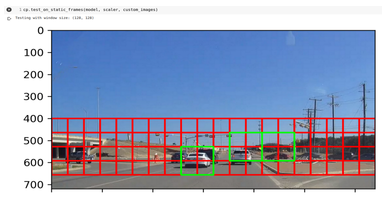


```
image = mpimg.imread("/content/road_p
plt.imshow(image)
```

```
marked_img = cv2.addWeighted(image, 0
plt.imshow(marked_img)
```

```
<matplotlib.image.AxesImage at 0x7f0e15cd1280><matplotlib.image.AxesImage at 0x7f0e15464cd0>
```

(a) Part of the lane detection activity

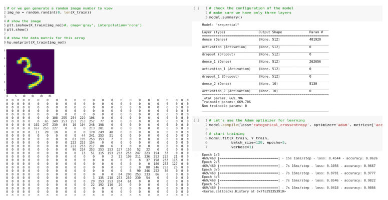


```
cp.test_on_static_frames(model, scaler, custom_images)
```

```
Testing with window size: (128, 128)
```

(b) Part of the vehicle detection activity.

Figure 3: Autonomous driving programming activities.



```
# or we gen generate a random image number to view
img_no = random.randint(0, len(X_train))

# show the image
plt.imshow(X_train[img_no])#, cmap='gray', interpolation='none')
plt.show()

# show the data matrix for this array
hp.matprint(X_train[img_no])
```

```
# check the configuration of the model
# make sure we have only three layers
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
activation (Activation)      (None, 512)               0
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
```

```
# Let's use the Adam optimizer for learning
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc

# start training
model.fit(X_train, Y_train,
          batch_size=128, epochs=5,
          verbose=1)
```

```
Epoch 1/5
469/469 [==============================] - 15s 16ms/step - loss: 0.4544 - accuracy: 0.8626
Epoch 2/5
469/469 [==============================] - 7s 16ms/step - loss: 0.1056 - accuracy: 0.9667
Epoch 3/5
469/469 [==============================] - 7s 16ms/step - loss: 0.0701 - accuracy: 0.9777
Epoch 4/5
469/469 [==============================] - 7s 16ms/step - loss: 0.0546 - accuracy: 0.9822
Epoch 5/5
469/469 [==============================] - 7s 16ms/step - loss: 0.0418 - accuracy: 0.9866
<keras.callbacks.History at 0x7fa293353910>
```

Figure 4: MNIST handwritten digit recognition.

classic digit recognition problem with the MNIST data set [29] (Figure 4). In this task, the students programmed following the example code to experience tasks such as data visualization, feature analysis, data normalization, neural network construction, model training, and model validation. The students were also asked to write code independently to test their models with random images.

2.3 Guest Speakers and Panel Discussion

Because the goal of *ExploreSTEM* is to provide STEM career experiences, we also invited guest speakers from fields outside software engineering and deep learning. Our guest speakers included architect, pilot, aircraft engineer, Geologist, Bio-chemist, electronic engineer, and computer graphics designer. Moreover, these guest speakers also provided personal career experiences that were missing from the lectures and hands-on activities, such as careers paths, motivations, work environments, and even personal struggles. One of our guest speakers also had disability.

In the last day of our 2021 camp, a panel was assembled to provide career and education experiences. The panel consisted of five panelists who were either STEM college/post-graduate students or professionals with disabilities. Each panelist first described their career/education experience. Then together, they answered the questions raised by the students, parents, and social workers.

3 OUTCOMES

3.1 Outcome of Programming and Lectures

Overall, the summer camps were successful. All students were able to finish all programming activities. The majority of them also finished the exercises with no or limited help. The capabilities of the students did vary. Some students may need help with debugging, which was easily addressed over Colab file sharing. Some students needed more time to program in our tutoring sessions. Nonetheless, several students could write programs as well as students without disabilities. One student with Autism could even finish the programming on first attempts without any bugs, which corroborates the belief that some with Asperger's Syndrome or high-functioning Autism may be well suited for software development [21].

We also employed post-camp surveys, which included questions on programming and machine learning concepts. Among the 15 students who attended in two years, 11 of them showed good understanding of most concepts in the survey. The students also viewed the camp positively, as exemplified in a news report [22].

These outcomes have two implications.

- Teaching virtual programming summer camp to students with disability is feasible. There need changes to the teaching and the material, and require assistance from special educators. However, the amount of effort required to make these changes is not significantly high.
- Our experience with this camp showed that students with disabilities, such as Autism, learning disabilities, and BVI, may become successful CS professionals. However, teaching CS students with disabilities previously in standard college classes was less successful for the authors. This difference suggests that there may be a need to redesign CS courses specifically for students with disabilities, a need to directly involve special educators.

3.2 Outcome of Guest Speakers and Panel

The guest speakers and panel had become a more important part of the summer camp than we initially expected. The personal STEM career experiences were easier for the students to understand, especially if the experience is also from someone with disabilities. The students were particularly interested in the speakers' careers path and their personal accounts of handling hardship in their work.

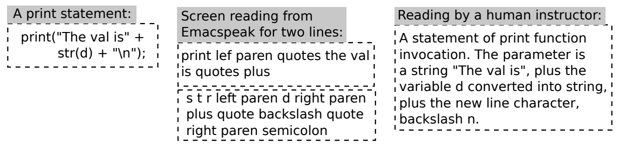
Moreover, several students also expressed interest in non-CS STEM fields, which were only discussed by the guests. Some of our students were eager to join STEM fields because they had innovative ideas that utilize STEM technologies to improve the life of people with disabilities. These ideas imply that students with disabilities joining STEM fields will not only benefit themselves but may also benefit the community of disabilities.

4 LESSONS LEARNED

4.1 Lessons for Curriculum Design

Programming Activity Adjustment. The programming and computer vision topics were interesting and manageable for the students. However, the programming activities did need adjustments.

Our programming activities were derived from summer camps for students without disabilities. After the first day, we learned that



A print statement:

```
print("The val is" +
      str(d) + "\n");
```

Screen reading from Emacspeak for two lines:

print lef paren quotes the val is quotes plus

s t r left paren d right paren plus quote backslash quote right paren semicolon

Reading by a human instructor:

A statement of print function invocation. The parameter is a string "The val is", plus the variable d converted into string, plus the new line character, backslash n.

Figure 5: Screen reader's reading and human's reading for a Python statement.

these activities had two problems. First, these activities required each student to type six to ten statements at a time. However, typing six to ten statements may introduced many typos, which were very stressful for the students with Autism to fix. Second, many statements mixed letters, digits, and punctuation marks, which made screen reading hard to understand.

Therefore, after the first day of our camp, we redesigned the programming activities so that each time a student only needed to type no more than five statements (usually 2 to 3). These statements cloud also be independently executed and examined, reducing the difficulty of debugging. These short chunk of executable statements also followed the "chunking" material principle in the evidence-based teaching practices [42]. To be more friendly to the screen reader, we also reduced the number of digits and punctuation marks, by offering wrapper functions with fewer parameters, and renaming the identifiers to use only common English words. These modifications turned out to be quite effective to enabling independent programming activities.

Schedule. Our schedules in 2020 and 2021 (Table 1 and Table 2) were different due to funding requirements. However, in 2021, we observed that the students were usually tired and less focused in the afternoon. Therefore, we recommend the instructions were only given in the mornings. The additional tutoring time was also very useful for handling students' different programming progresses.

Non-CS Topics. Including guest speakers from non-CS fields complemented the summer camp and helped students who are interested in other STEM fields. Moreover, a particularly interesting observation from the panel discussion was the lack of information on financial support for college education. Several students and parents asked our panelists about the financial means to support their education. The panelists provided some federal and state resources. These resources were little known to the students and their parents. This observation made us believe that STEM out-reaching programs for students with disabilities should also include some discussion on the financial support resources.

4.2 Lessons for Technology and Accessibility

For most students, Google Colab was sufficient. Occasionally, the instructors and tutors would use Colab's share function to directly modify a student's code for difficult bugs. This debugging support through Colab sharing also follows the "immediate, corrective-feedback" principle in the evidence-based practice for teaching [36, 42]. Overall, our experience with Google Colab is positive.

However, we did encounter three accessibility difficulties for the student with blindness. The first difficulty is the screen reader's limited capability to properly read computer program statements. Our student used the Jaws screen reader [18]. However, screen readers are designed to read English text, not program statements, which are usually mixtures of English letters, digits, punctuation marks, and white spaces. Therefore, when applied to program statements, screen readers usually deteriorate to read one character at a time, making it very difficult for the listeners to understand. Figure 5 gives an example of how a screen reader reads a Python statement, where every character is read except for a few English words. Even with the adjusted programming activities, the reading could still be confusing, and our instructors had to read the statements instead. Figure 5 also shows how our instructor read this statement, where the meaning of the statement is read instead of just characters. Although the instructor still read individual characters occasionally to teach the syntax, reading the statement by its meaning makes programming education easier to students with blindness.

The second difficulty comes from code navigation. Programming involves frequent cursor moving (navigation) within program code to fix bugs. This common task, however, was very difficult for our student with blindness (and also for programmers with blindness [4]). To find a specific statement, the student needed to move the cursor line by line, while listening to the screen readings to determine if they have hit the statement they want to visit. This process was so time-consuming and frustrating that, in every case, the student gave up debugging and asked the instructors for help. Our instructors usually had to tell the student how many lines to move to locate the desired statement.

Although we addressed the above two accessibility issues with human efforts, ideally, better screen reading and better support for code navigation, should be provided to aid students with blindness and vision impairment (BVI).

The third accessibility difficult comes from the teaching material. As we taught Computer Vision, many of our activities involved images. Initially, we found the images were ignored by the screen reader. We assumed screen readers couldn't handle images, and we described the image to the students. However, later we found that the screen reader was able to describe some images from Udacity's data set (likely due to Jaws' Picture Smart database), which made learning much easier for the student. Therefore, we recommend adding image descriptions in the teaching material of all image outputs to make them more accessible for screen readers. Note that the student with blindness was indeed interested in Computer Vision, as he saw it as a necessary tool to assist people with BVI.

4.3 Lessons for Special Educator Involvement

It is necessary that special educators present at the camp to monitor the class and intervene when necessary. The special education professor was indispensable to our camp in four aspects.

First, the special education professor provided training to our CS professors and tutors, which included characteristics of disabilities, accommodations for learning, and strategies that promote engagement. The training also led the instructors to follow the evidence-based practice for teaching students with disabilities [17, 36, 42], such as explicit instructions, clear communications, chunking materials in logical orders, and corrective feedback.

Second, the CS professors might still overlook certain teaching requirements even after being trained. The special education professor would intervene to clarify or correct the instruction. For example, when teaching students with Autism, the instructions much be clear and specific. However, this requirement tended to be forgotten during teaching. There was once, at the end of the day, the CS professor tried to dismiss the class by saying "That's all for today, and I'll see you tomorrow." This sentence implied that the students should leave the online meeting. However, some students with Autism didnot understand the implied message and remained in the meeting. Our special educator immediately discovered the issue and told the students that they should leave.

Third, our special education professors were more capable at discovering when students were over-stressed and calming them down. For example, some of our students got stressed if there were many bugs in their code. Our special education professor would intervene in this case, told the CS instructors to temporarily leave the student, and calmed down the student before proceeding to further debugging. Our special education professor was constantly monitoring the class to handle similar issues.

Forth, the special education professor was better at handling the large variety of disabilities. Our summer camp was designed for students with any disabilities. Even for the same disability, different students manifest differently. It was impossible that pre-camp training could enable the CS instructors to handle every issue. Hence, the presence of special educators is a must for us.

5 RELATED WORK

AccessCSforAll [25] is a joint research effort by University of Washington and University of Nevada Las Vegas to offer CS education to K-12 students with disabilities. AccessCSforAll provided novel programming language [51, 52], accessibility tools, and curricula. University of Washington also had a program called DO-IT [37], which aims at assisting people with disability in education, research, and career. Our summer camp is partially inspired by these prior efforts. DO-IT also directly assisted us with the panel discussion. CS education for students with disabilities has recently received more attention from CS educators. SIGCSE has held keynote talk [13], panels [24, 27] and Birds-of-a-Feather sessions [1, 26] for CS educators who are interested in educating students with disabilities. Game-of-Life is a cellular automata framework for CS outreach [28]. Blaser and Ladner also reported and analyzed the difficulty to collect and understand data on disability in CS education [8]. We hope our report could also provide useful experience and lessons to like-minded educators.

The difficulties of teaching CS to individuals with BVI were observed by prior studies [5, 6, 16, 39, 50]. Emacspeak [41] allows reading code in the Emacs editor. JavaSpeak [11, 48] was based on Emacspeak to read Java programs. Besides speech, auditory cues, such as spearcons and white noses, have been shown to be effective in assisting programmers with BVI [3, 31, 49]. These accessibility tools, however, are not available for Google Colab.

Many studies also observed the challenge of teaching programming to students with BVI due to the complex syntax [32, 38]. Quorum [52] is a text-based programming language that has more intuitive statements. Kane and Bigham [19] reported that Ruby might be a more suitable language for students with BVI. Audio Programming Language (APL) [43] was a programming language with a set of reduced (simplified) commands. Due to the popularity of block-based languages (BBL), many studies also investigated making BBL more accessible [35], such as Bonk [20], StoryBlocks [23], and Block4All [33]. Physical programming devices have also been developed. Ludi et al. taught programming through robotics [30]. Torino [34] included a new language and physical programming devices with beads and cables. Howard et al. employed haptic and auditory tools to teach programming with Lego Mindstorms [14]. Franqueiro and Siegfried improved Visual Basic to make it more accessible [12]. Schanzer et al. investigated improving WeScheme for students with BVI, which is a web-based programming environment for language Scheme [45]. These new languages and devices are powerful at introducing computer programming and its concepts. However, to provide a career experience, we taught Python in our summer camp, which is more common in the IT industry.

There was also work for better code navigation and debugging. CodeTalk [40] was a plugin for Visual Studio that offers code summary trees, code summaries, and function lists. StructJumper [7] translated nested Java classes into a tree to simplify code reading. Smith et al. [47] improved tree views for code navigation. Schanzer et al. [44]. employed compiler syntax analysis to aid in navigating nested hierarchical code. These tools may potentially be integrated into Colab or Juypter Notebook to improve accessibility.

6 CONCLUSIONS

This paper presented our experience in teaching an online programming and machine learning summer camp for high-school students with disabilities. Overall, our experience shows that online programming camps for high-school students with disabilities are feasible, and students with disabilities can potentially be successful Computer Science professionals. Nonetheless, accommodations and adjustments were necessary, especially in the curriculum, schedule, teaching practices, and the involvement of special educators. We plan to continue offer this camp in the future.

ACKNOWLEDGEMENT

This work is primarily supported by Texas Workforce Commission (TWC) Vocational Rehabilitation Services contracts 3020VRS072 and 3021VRS085. Wei Wang and Kathy B. Ewoldt were also partially supported by the National Science Foundation (NSF) under grant 2202632. Wei Wang was also partially supported by NSF grants, 2221843, 2155096, and 2215359. Alberto M. Mestas-Nuñez was supported in part by NASA Grant 80NSSC19M0194. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of TWC, NSF, and NASA. We thank Tammy Winkenwerder, Kevin Markel, Adriana Martinez, Mayra Gutierrez, and all TWC staff involved for the camp organization. We would also like to thank our guest speakers and panelists, as well as Kayla D. Brown and University of Washington's DO-IT program for organizing the panel. The authors sincerely thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] Jennfier Akullian, Adam Blank, Brianna Blaser, Elba Garza, Christian Murphy, and Kendra Walther. 2022. Diversity Includes Disability Includes Mental Illness: Expanding the Scope of DEI Efforts in Computer Science. In *Proc. of the 53rd ACM Technical Symp. on Computer Science Education V. 2.*

[2] Abdulmalek Al-Gahmi, Yong Zhang, and Hugo Valle. 2022. Jupyter in the Classroom: An Experience Report. In *ACM Technical Symp. on Computer Science Education.*

[3] Khaled Albusays. 2018. Exploring Auditory Cues to Locate Code Errors and Enhance Navigation for Developers Who Are Blind. *SIGACCESS Access. Comput.* 120 (Jan. 2018), 11–15.

[4] Khaled Albusays, Stephanie Ludi, and Matt Huenerfauth. 2017. Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges. In *Int'l ACM SIGACCESS Conference on Computers and Accessibility.*

[5] Ameer Armaly, Paige Rodeghero, and Collin McMillan. 2018. A Comparison of Program Comprehension Strategies by Blind and Sighted Programmers. *IEEE Trans. on Software Engineering* 44, 8 (2018), 712–724.

[6] Catherine M. Baker, Cynthia L. Bennett, and Richard E. Ladner. 2019. Educational Experiences of Blind Programmers. In *ACM Technical Symp. on Computer Science Education.* 759–765.

[7] Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. 2015. StructJumper: A Tool to Help Blind Programmers Navigate and Understand the Structure of Code. In *Proc. of ACM Conf. on Human Factors in Computing Systems.*

[8] Brianna Blaser and Richard E. Ladner. 2020. Why is Data on Disability so Hard to Collect and Understand?. In *Research on Equity and Sustained Participation in Engineering, Computing, and Technology.*

[9] Kevin Buffardi. 2017. Comparing Remote and Co-Located Interaction in Free and Open Source Software Engineering Projects. In *ACM Conf. on Innovation and Technology in Computer Science Education.*

[10] Matthias Drusch, Umberto Del Bello, Sébastien Carlier, Olivier Colin, Veronica Fernandez, Ferran Gascon, Bianca Hoersch, Claudia Isola, Paolo Laberinti, Philippe Martimort, et al. 2012. Sentinel-2: ESA's optical high-resolution mission for GMES operational services. *Remote sensing of Environment* 120 (2012), 25–36.

[11] Joan M. Francioni and Ann C. Smith. 2002. Computer Science Accessibility for Students with Visual Disabilities. In *SIGCSE Technical Symp. on Computer Science Education.*

[12] Kenneth G. Franqueiro and Robert M. Siegfried. 2006. Designing a Scripting Language to Help the Blind Program Visually. In *Int'l ACM SIGACCESS Conference on Computers and Accessibility.*

[13] Vicki L. Hanson. 2007. Inclusive Thinking in Computer Science Education. In *Annual SIGCSE Conf. on Innovation and Technology in Computer Science Education.*

[14] Ayanna M. Howard, Chung Hyuk Park, and Sekou Remy. 2012. Using Haptic and Auditory Interaction Tools to Engage Students with Visual Impairments in Robot Programming Activities. *IEEE Trans. on Learning Technologies* 5, 1 (2012), 87–95.

[15] Earl W. Huff, Kwajo Boateng, Makayla Moster, Paige Rodeghero, and Julian Brinkley. 2020. Examining The Work Experience of Programmers with Visual Impairments. In *IEEE Int'l Conference on Software Maintenance and Evolution.*

[16] Earl W. Huff, Kwajo Boateng, Makayla Moster, Paige Rodeghero, and Julian Brinkley. 2021. Exploring the Perspectives of Teachers of the Visually Impaired Regarding Accessible K12 Computing Education. In *Proc. of ACM Technical Symp. on Computer Science Education.*

[17] Charles A. Hughes, Jared R. Morris, William J. Therrien, and Sarah K. Benson. 2017. Explicit Instruction: Historical and Contemporary Contexts. *Learning Disabilities Research & Practice* 32, 3 (2017), 140–148.

[18] Freedom Scientific Inc. 2022. JAWS. https://www.freedomscientific.com/products/software/jaws/. (2022).

[19] Shaun K. Kane and Jeffrey P. Bigham. 2014. Tracking @stemxcomet: Teaching Programming to Blind Students via 3D Printing, Crisis Management, and Twitter. In *ACM Technical Symp. on Computer Science Education.*

[20] Shaun K. Kane, Varsha Koushik, and Annika Muehlbradt. 2018. Bonk: Accessible Programming for Accessible Audio Games. In *ACM Conf. on Interaction Design and Children.*

[21] L. R Kendall. 2013. *A Phenomenological Inquiry into the Perceptions of Software Professionals on the Asperger's Syndrome/High Functioning Autism Spectrum and the Success of Software Development Projects.* Ph.D. Dissertation. Capella University.

[22] KENS-TV. UTSA camp expands STEM opportunities for teens with disabilities. https://www.kens5.com/article/news/education/coding-classes-utsa/273-ff650357-fc44-4724-b01a-69a36db4a5ae. (????).

[23] Varsha Koushik, Darren Guinness, and Shaun K. Kane. 2019. StoryBlocks: A Tangible Programming Game To Create Accessible Audio Stories. In *CHI Conf. on Human Factors in Computing Systems.*

[24] Richard E. Ladner, Caitlyn Seim, Ather Sharif, Naba Rizvi, and Abraham Glasser. 2021. Experiences of Computing Students with Disabilities. In *Proc. of ACM Technical Symp. on Computer Science Education.*

[25] Richard E. Ladner, Andreas Stefik, and Brianna Blaser. 2019. Addressing Disability in CS for All. In *Research on Equity and Sustained Participation in Engineering, Computing, and Technology.*

[26] Richard E. Ladner, Andreas Stefik, Amy J. Ko, and Brianna Blaser. 2019. Access to Computing Education for Students with Disabilities. In *ACM Technical Symp. on Computer Science Education.* 1249.

[27] Richard E. Ladner, Andreas Stefik, Amy J. Ko, Brianna Blaser, Stacy Branham, and Raja Kushalnagar. 2022. Disability in Computer Science Education. In *ACM Technical Symp. on Computer Science Education V. 2.*

[28] Richard E. Ladner and Tammy VanDeGrift. 2008. The Game of Life: An Outreach Model for High School Students with Disabilities. *SIGCSE Bull.* 40, 1 (mar 2008).

[29] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (1998).

[30] Stephanie Ludi, Lindsey Ellis, and Scott Jordan. 2014. An Accessible Robotics Programming Environment for Visually Impaired Users. In *Int'l ACM SIGACCESS Conference on Computers & Accessibility.*

[31] Stephanie Ludi, Jamie Simpson, and Wil Merchant. 2016. Exploration of the Use of Auditory Cues in Code Comprehension and Navigation for Individuals with Visual Impairments in a Visual Programming Environment. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility.*

[32] S. Mealin and E. Murphy-Hill. 2012. An Exploratory Study of Blind Software Developers. In *IEEE Symp. on Visual Languages and Human-Centric Computing.*

[33] Lauren R. Milne and Richard E. Ladner. 2018. Blocks4All: Overcoming Accessibility Barriers to Blocks Programming for Children with Visual Impairments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.*

[34] Cecily Morrison, Nicolas Villar, Anja Thieme, Zahra Ashktorab, Eloise Taysom, Oscar Salandin, Daniel Cletheroe, Greg Saul, Alan F Blackwell, Darren Edge, Martin Grayson, and Haiyan Zhang. 2020. Torino: A Tangible Programming Language Inclusive of Children with Visual Disabilities. *Human–Computer Interaction* 35, 3 (2020), 191–239.

[35] Aboubakar Mountapmbeme and Stephanie Ludi. 2020. Investigating Challenges Faced by Learners with Visual Impairments Using Block-Based Programming/Hybrid Environments. In *Int'l ACM SIGACCESS Conference on Computers and Accessibility.*

[36] Institiue of Education Sciences. 2022. Evidence-based teaching practices. https://ies.ed.gov/ncee/edlabs/infographics/pdf/REL_SE_Evidence-based_teaching_practices.pdf. (2022).

[37] University of Washington. 2022. DO-IT: Disabilities, Opportunities, Internetworking, and Technology. https://www.washington.edu/doit/. (2022).

[38] Charles B Owen, Sarah Coburn, and Jordyn Castor. 2014. Teaching Modern Object-Oriented Programming to the Blind: An Instructor and Student Experience. In *2014 ASEE Annual Conference & Exposition.* 24–1167.

[39] Maulishree Pandey, Vaishnav Kameswaran, Hrishikesh V. Rao, Sile O'Modhrain, and Steve Oney. 2021. Understanding Accessibility and Collaboration in Programming for People with Visual Impairments. *Proc. ACM Hum.-Comput. Interact.* 5 (April 2021).

[40] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y. Vidya, Manohar Swaminathan, and Gopal Srinivasa. 2018. CodeTalk: Improving Programming Environment Accessibility for Visually Impaired Developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.*

[41] T.V. Raman. 1997. Emacspeak-an Audio Desktop. In *IEEE COMPCON.*

[42] Barak Rosenshine. 2012. Principles of Instruction: Research-based Strategies that All Teachers Should Know. *American Educator* 36, 1 (2012), 12.

[43] Jaime Sánchez and Fernando Aguayo. 2005. Blind Learners Programming through Audio. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems.*

[44] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. 2019. Accessible AST-Based Programming for Visually-Impaired Programmers. In *ACM Technical Symp. on Computer Science Education.*

[45] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. 2020. Adapting Student IDEs for Blind Programmers. In *Koli Calling Int'l Conf. on Computing Education Research.*

[46] Darja Šmite, Nils Brede Moe, and Richard Torkar. 2008. Pitfalls in Remote Team Coordination: Lessons Learned from a Case Study. In *Product-Focused Software Process Improvement*, Andreas Jedlitschka and Outi Salo (Eds.).

[47] Ann C. Smith, Justin S. Cook, Joan M. Francioni, Asif Hossain, Mohd Anwar, and M. Fayezur Rahman. 2003. Nonvisual Tool for Navigating Hierarchical Structures. In *Proc. of Int'l ACM SIGACCESS Conf on Computers and Accessibility.*

[48] A. C. Smith, J. M. Francioni, and S. D. Matzek. 2000. A Java Programming Tool for Students with Visual Disabilities. In *Int'l ACM Conf. on Assistive Technologies.*

[49] Andreas Stefik, Christopher Hundhausen, and Robert Patterson. 2011. An Empirical Investigation into the Design of Auditory Cues to Enhance Computer Program Comprehension. *Int'l Journal of Human-Computer Studies* 69, 12 (2011).

[50] Andreas Stefik, Richard E. Ladner, William Allee, and Sean Mealin. 2019. Computer Science Principles for Teachers of Blind and Visually Impaired Students. In *ACM Technical Symp. on Computer Science Education.*

[51] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *ACM Trans. Comput. Educ.* (Nov. 2013).

[52] Andreas M. Stefik, Christopher Hundhausen, and Derrick Smith. 2011. On the Design of an Educational Infrastructure for the Blind and Visually Impaired in Computer Science. In *ACM Technical Symp. on Computer Science Education.*

[53] Auro Robotics Udacity. 2022. An Open Source Self-Driving Car. https://github.com/udacity/self-driving-car. (2022).

[54] Maria Wachal. 2019. What is it like to work remotely as a software developer? https://blog.softwaremill.com/what-is-it-like-to-work-remotely-as-a-software-developer-1c0777e4a2a9. (2019).