This project report is provided as a reference, not as a standard. Some comments are included to guide students. Please keep the following in mind:

- The project report should be concise, focusing on the work completed by the group. Excessive use of ChatGPT with irrelevant details will result in mark deductions.
- The introduction and other details should be brief and to the point. Abstract and conclusion should be well thought. Students should refer to the rubric for clarification regarding all marking criteria.

Abstract has more background and less details about problem statement, procedure and key conclusion.
Testing details missing in the abstract.

Abstract: Our society's functioning depends on visual cues and our understanding of them. Most of the information presented to us in our environment, including bus stop timetables, road crossings, and caution signs, are all visual. However, an estimated 2.2 billion people are visually impaired worldwide. Additionally, since age is a leading cause of vision loss, this number is also expected to increase as the average lifespan increases. This report will introduce a device developed to assist in the lives of the visually impaired. We present a handbag with a camera, microphone, and earphones, which may assist visually impaired people in answering questions about their environment and finding objects. Users can press a button, give a verbal prompt, and receive an audio response from the device with a response to their prompt. They could ask the device for a description of their environment or for specific information, such as the colour of an object. The device also contains an option to help find an object, in which the device will continuously guide the user towards the object. The device was created using a Raspberry Pi that records audio and images and sends them through the internet to a server for processing using modern machine-learning models. We hope this device will act as an effective prototype to promote the development of purpose-built devices for the visually impaired.

Keywords: Navigation, Python, Raspberry Pi, Speech Commands, Internet of Things

1 INTRODUCTION

The functioning of our society is dependent on visual cues and our understanding of them. Navigation information such as bus stop signs, road crossings, and caution signs rely heavily on vision. This can lead to many challenges for the visually impaired when attempting to do tasks that many of us would take for granted, such as taking public transport [8]. Additionally, many of the activities required in our lives rely on vision, such as finding groceries or clothes to buy, cooking, and cleaning. For visually impaired people, the information required for these tasks is not easily accessible, which hinders their ability to navigate our increasingly visual world [6, 8]. The World Health Organization estimated that 2.2 billion people have visual impairments worldwide [4]. Additionally, with our aging population, this number is expected to rise as age is the leading cause of visual impairment. Therefore, the problem of navigating our world with visual impairments is a problem that will affect a large number of people, and their friends and family. Thankfully, glasses can assist many to overcome their visual impairment, but this is not not an option for all visually impaired people. ✓ ✓

The loss or complete deterioration of existing vision can be very frightening and overwhelming, leaving those affected to worry about their ability to maintain their independence and support themselves and their family. The visually impaired person may needs help to do even small chores in their day-to-day life. Previous work has demonstrated the advantages of using wearable IoT devices to assist the visually impaired in their daily tasks [1]. Adep et al. demonstrated the use of a Raspberry Pi device to detect when users ✓

Introduction is of appropriate length.

moved their hand close to hot objects, so that they could cause a vibration to deter the visually impaired person from burning themselves [1].

The marked text on right (within red line) should be in technical content.

This report will present a wearable internet of things (IoT) device, powered by recent advances in machine learning for language and image recognition, that can help visually impaired people to understand the world around them. The device aims to assist in the identification of small visual cues, to help the user experience the world around them, and to find objects. Traditionally, understanding the world around us in a general way was an intractable problem for computer systems. However, in the previous decade there have been incredible advancements in the use of machine-learning systems to identify objects in images, and to answer questions about images [2, 7, 11]. Additionally, there have recently been great advances in the use of voice recognition to understand verbal prompts [10]. These major advancements lend themselves well to the creation of a device to use the visual understanding of machine learning systems, and the audio understanding of voice recognition systems, to assist visually-impaired users in their day-to-day lives. ✓ ✓

The device presented in this report was created as a wearable handbag, with a camera to record the surroundings of the user, a microphone to record voice prompts, and a set of earphones to speak audio responses to the user. The device has a button that will trigger a voice recording when pressed. The user may then speak their question, which will be recoreded. A photo of their environment will then be taken. The start and end of the voice recording is indicated by audible ping noises to notify the user when to ask their question. The voice recording and the photo are then sent to a server for processing using state-of-the-art machine learning models. The voice recording will be analysed using OpenAI Whisper, which is a state-of-the-art voice-recognition software released in September, 2022 by the OpenAI research lab [10]. It achieves world-class performance at transcribing speech into text, and our device utilises this to accurately recognise user prompts. An analysis of the prompt from the user is then performed to identify the type of prompt that the user has asked, so that a response can be generated. ✓ ✓

Once the prompt has been analysed, one of two state-of-the-art machine learning models for visual question answering will be used to answer questions and identify objects in the image [11]. Two OFA models are used for this task, which are state-of-the-art models for visual-question answering that were released in February, 2022. Visual question answering involves the answering of a question about an image. For example, users may ask for the colour of a shirt, or the location of an object, and the models will be able to answer these questions. These models are still not perfect, but they are very capable at answering simple questions about images. ✓

The combination of our presented wearable IoT device, alongside the state-of-the-art machine learning models, allows users to ask general questions about their surroundings, and get automated responses to their questions in English. This will hopefully lead to greater autonomy in the lives of visually impaired people when using the proposed device. ✓

Problem statement should be given explicitly.
At the end of the introduction, give the preview of the report (stating what section describes what).

2 DESIGN

One of the most valuable things for a disabled person is to gain independence in their ability to do their day-to-day activities. Devices designed specifically for disabled people can assist them to lead more independent lives. The original design of our system intended to focus on the creation of a device to assist users in their purchasing of groceries. However, we later discovered that this task has already had a lot of research attention [5, 12]. Therefore, we instead pivoted to implement the state-of-the-art visual question answering models into our device. Visual question answering (VQA) allows the users to ask arbitrary questions about a photo, and the VQA machine-learning model will produce a text response answering the question. Therefore, using these machine-learning models, we can create an assistant to assist in visually impaired people's understanding of their surroundings. The device has the opportunity to assist visually impaired people in answering simple questions that they have about their environment that other, more specialised, visual assistant tools cannot answer [5, 12]. It was identified that a system to facilitate the use of these visual question answering models would require the following steps, ✓ ✓

Extra text is given. These seven points can be only bullet points or very short explanation, only if necessary.

(1) **The user will have to indicate to the system that they wish to ask a question**. Many other voice-based systems use wake words for this task [3]. Wake words are simple verbal commands, often single words, that will indicate to a system that it should wake to receive a command. For example, Apple's Siri system uses the wake word "Hey, Siri". These types of wake words provide a more natural way of interacting with systems, but they requires that a device is always listening for the wake word. In our case, the implementation of wake words was not feasible, as the general speech detection systems we tested were too slow or innaccurate to monitor the microphone in real-time. Therefore, we chose to implement a button using the GPIO pins of the Raspberry Pi that will activate the system when it is pressed. ✓
(2) **The system will have to record the user's verbal prompt**. Once the user has indicated that they wish to ask a question, the system will then have to record the audio input from the microphone while the user speaks. The end of a user's prompt may be detected based upon the volume of the recorded audio from the microphone. When the volume is low for 2 seconds, the device will consider the prompt to be finished. ✓
(3) **The system will have to take a photo of the user's environment**. Once the user has finished asking their question, the system will have to take a photo to capture the user's environment. This photo will be used by the visual question answering model to detect features of the user's surroundings. ✓
(4) **The system will have to transcribe the verbal prompt to text**. It is difficult for us to analyse verbal prompts directly from the audio. However, by transcribing the audio to text, we can use traditional text parsing techniques to identify the intent of the user. The audio transcription is performed on the server, due to the limited processing power of the Raspberry Pi. ✓
(5) **The text prompt should be parsed for intent**. We have three built-in modes of operation based upon the prompt that the user asked. If the user asks a question, then the system will attempt to answer

✓

the question. If the user asks the system to ”describe“, then the system will generate a description of the photo for the user. If the user asks the system to ”find“ an object, then the system will attempt to locate the object, and inform the user about its location. ✓ ✓

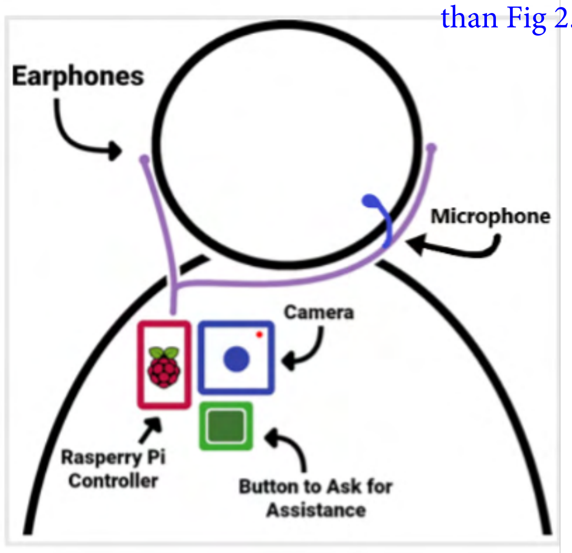
(6) **The machine-learning models should be invoked to answer questions about the photo**. The OFA model for visual-question answering, or the OFA model for object grounding, should be invoked with the photo and the text prompt [11]. This will allow the model to generate an answer to the question that the user asked. However, the object grounding model has an issue with detecting when an object is missing. Therefore, when detecting the location of an object, we must first invoke the visual-question answering model for it to detect if the object is in the image. ✓

(7) **The response should be spoken to the user**. The response generated from the OFA models should be spoken to the user to inform them about the answer to their question. If the user issued a find command, then the system will first describe the location of the object in the image in text, to speak to the user. The description of the location of an object is done based upon the size of the bounding-box around the object, and the relative location of the centre of the bounding-box in the image. For example, a user may be informed that an object is ”moderately sized, and to the left“. The intent is that users will be able to learn to interpret these descriptions based upon the size of the object they are looking for. ✓

This process is shown in the flowchart shown in Fig. 2. An initial design concept was also developed to represent how a user may wear this device, as shown in Fig. 1.

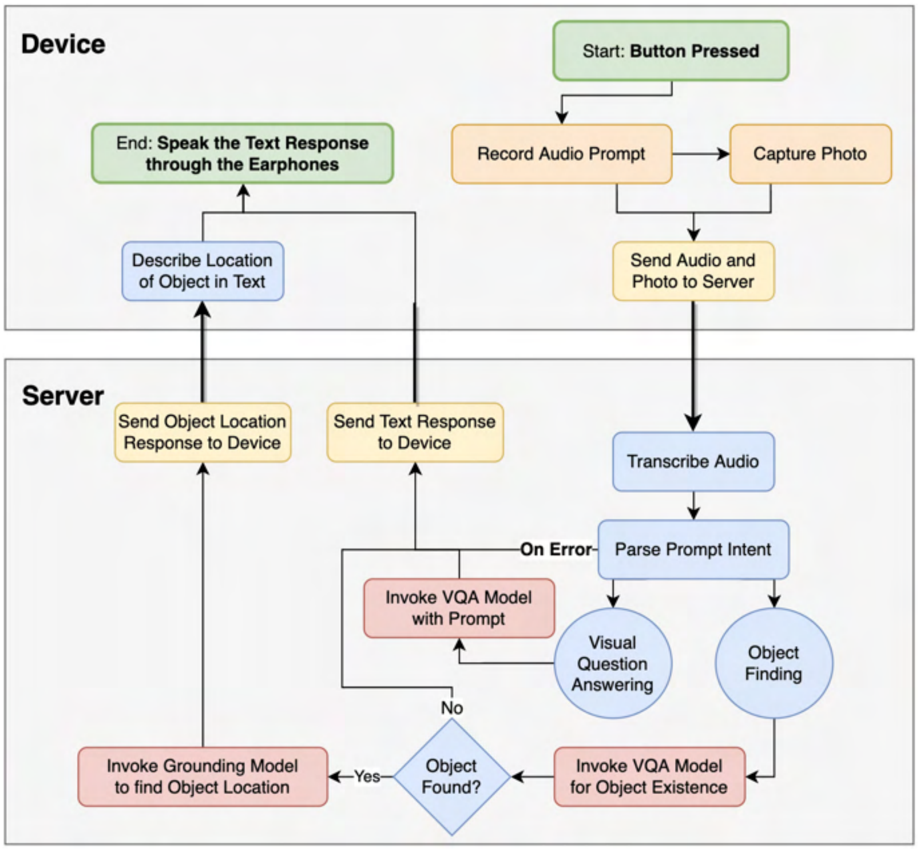
Fig 1 should be referred in text earlier than Fig 2.

Good effort with the diagram, but a basic block diagram would have been more meaningful.



✓

Fig. 1. An initial design concept for the device.


Device
Start: Button Pressed
Record Audio Prompt
Capture Photo
Send Audio and Photo to Server
End: Speak the Text Response through the Earphones
Describe Location of Object in Text
Server
Send Object Location Response to Device
Send Text Response to Device
Transcribe Audio
Parse Prompt Intent
On Error
Invoke VQA Model with Prompt
Visual Question Answering
Object Finding
No
Object Found?
Yes
Invoke Grounding Model to find Object Location
Invoke VQA Model for Object Existence

✓

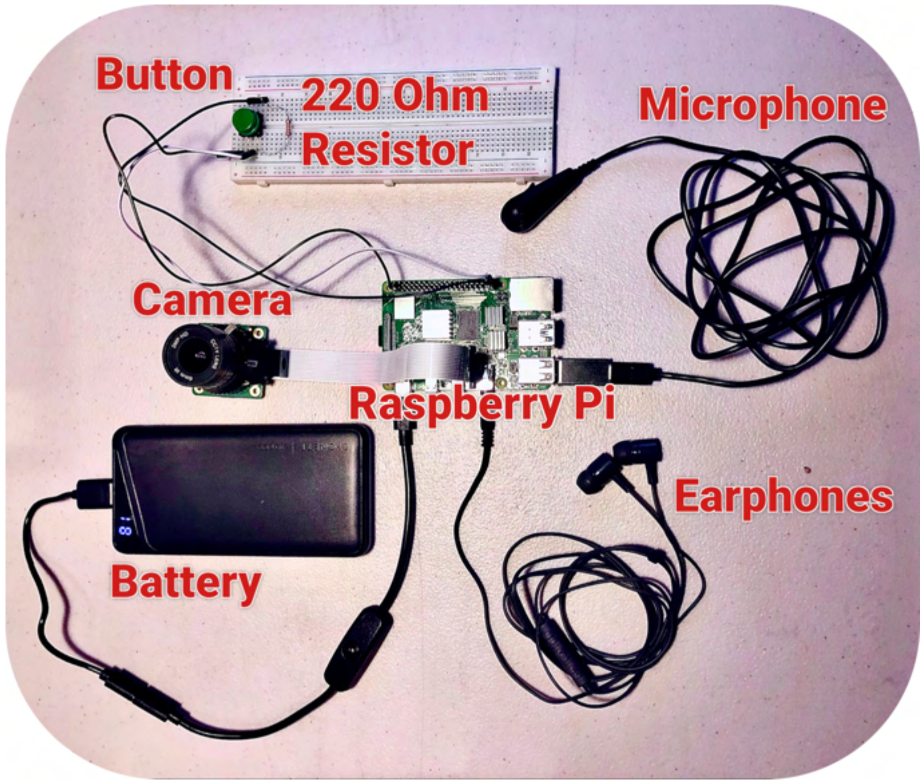
Fig. 2. A flowchart of the operation of the device from button press to speaking a response to the user.

3 HARDWARE

The device was implemented using several off-the-shelf electrical components combined with a Raspberry Pi. A Raspberry Pi 4b was used as the microprocessor for the device, to which all the peripherals were connected. ✓ An Audio-Technica clip-on microphone is used to detect the voice prompts from the user. A pair of cheap wired earphones are used to speak the responses of the device back to the user. A Raspberry Pi high-quality camera, together with a 6mm wide-angle lens, was used to capture the images of the user's surroundings. A Cygnett Boost V2 10000mAh power bank is used to provide the power to the device. A small button component is used to receive input from the user, with three wires and a 220-ohm resistor to create its circuit. The hardware components that make up the device are shown in Fig. 3. ✓

3.1 Detecting Button Press

A simple circuit is used to detect the button presses using the Raspberry Pi's GPIO pins. The 3.3V pin of the Raspberry Pi is connected to one side of the button, and the GPIO pin is connected to the other side of the ✓



✓

Fig. 3. A photo of the hardware components used by the device, labelled in red.

button. The detection side of the button is then connected to a 220-ohm resistor to ground. This allows the button to be detected as HIGH when it is pressed, and LOW when it is not pressed.

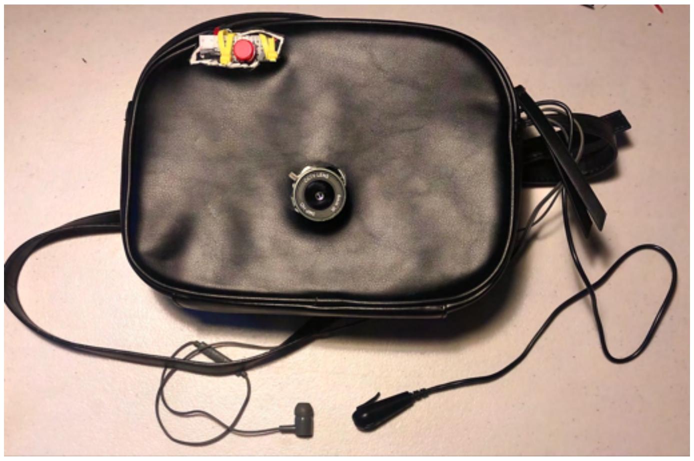
3.2 Server

The webserver to run the machine-learning models was hosted on a Macbook Pro from 2021. The machine-learning models were run on this laptop instead of on the Raspberry Pi due to its computation limits. The Raspberry Pi was not fast enough to compute the responses using these machine-learning models within a reasonable response time. Both the Whisper and the OFA models take significant computation time (in the order of seconds) to compute their responses [10, 11]. Therefore, we instead chose to run these models on a laptop. The requests from the Raspberry Pi are forwarded to the laptop through the internet. ✓

3.3 Assembly

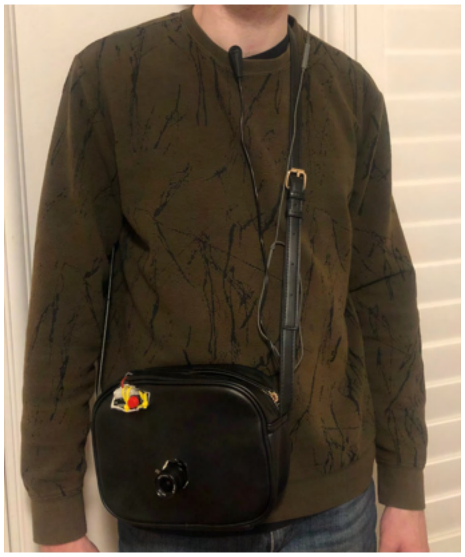
The device's hardware components were assembled into a handbag. Holes were cut into the handbag for the button and the camera. Zip ties are used to hold the button in-place, and a pocket within the handbag is used to hold the camera in-place. A photo of the device can be seen in Fig. 4. This assembly of the device ✓

allows it to be worn over the shoulder, as shown in Fig. 5. The assembly of the device into a bag was chosen to make the device easy to wear, and to allow the device to be less obvious to other people. This choice was made to respect the psychology of particular groups that do not want to be noticed by others. The handbag design was chosen in the attempt to make the bag more inconspicuous.



✓

Fig. 4. A photo of the device when assembled.



✓

Fig. 5. A photo of the device when worn.

4 SOFTWARE

The software on the device and on the server were both implemented using the Python programming language [9]. The Python programming language has good support on the Raspberry Pi and on Mac systems, and it is an easy language to use. This made it a good choice for our device. One of the main advantages of using Python is the wide availability of libraries that are available for it [9]. The full source-code for the software that is ran on the Raspberry Pi device, and to run the webserver to receive requests from the Rasperry Pi on the server, are given in the Appendices. ✓

4.1 Device Software

Our project made use of the *pyaudio* library to record the sound from the microphone. The *pydub* library was also used to play the notification sounds for users. The *pyttsx*3 library was used to speak text into the earphones for users of the device. The *picamera*2 and *OpenCV* libraries were used to capture and save photos to the device.

4.2 Server Software

The server-side of our project primarily made use of the Whisper and OFA libraries to perform the primary functions of transcribing the audio into text, and answering the questions about the photos taken using the device. The library *Flask* was also used to create a webserver for the device to send requests to. This project created modified versions of the OFA model code, which used the Gradio library to deploy the OFA models to a webserver on the laptop. The primary webserver sent requests to the webservers that hosted the machine-learning models when answering visual questions, or when grounding the location of objects in the image. Additional changes were made to the grounding model to return the location of the bounding boxes, instead of annotating them in the images. ✓ ✓

5 EVALUATION

The evaluation of the device was performed by testing its use for describing images, asking questions, and for finding objects. Better to write positive and negative evaluation under separate headings.

The image descriptions given by the device were effective in giving the context of a user's surroundings. However, often, the descriptions would focus on particular aspects of the photo instead of describing the whole scene. For example, in a picture with a person sitting at a desk, the system described the image as "a picture of a man with a beard". This is still helpful, but requires more detail to be useful in real-life. Additionally, sometimes the model would output more text that was not relevant to the image, such as "a picture of a man with a beard (colourised)". This is likely due to the fact that the training dataset used for the OFA models included many images from the internet, and not just photos and descriptions from real-life [11]. ✓

The answers of the system to visual questions was very effective for simple queries. For example, asking the system to identify the colour of pieces of clothing was quite reliable. However, the system struggled with other simple queries such as counting the number of objects on a table. The system would often respond ”many“ instead of a specific number. However, when the system did give a number, it was also often slightly off from the true count. An example photo taken using the device is given in Fig. 6. When the system was asked ”what is the subject of this photo¿‘, it answered ”a laptop“. This demonstrates one case where the system gave a reasonable and accurate response to a simple question.

✓

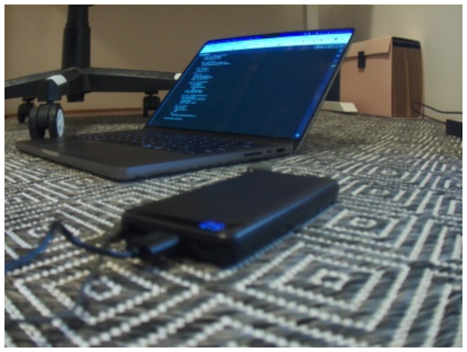


Fig. 6. An example photo taken using the device.

The system did not perform as well when attempting to help users find objects. The main issue of the system for this task is the delay between the user's movement and receiving an answer back from the server with information about where the object is in relation to the last photo that was taken. This could easily lead to the instructions being many seconds out-of-date. This problem was exasperated by the need to run two queries when finding objects. One query asked whether the object was in the image, and the next asked where the object was in the image. This need for answering two questions led the delay for finding objects to be much longer than for the question answering. However, for simple cases such as finding a chair, the system is able to guide a user to their destination if the user moves slowly. In the case that a visually impaired person was not able to find an object at all, this would present an improvement for the visually impaired person in their life. However, future work could likely see large improvements in this area of the system. ✓

Some quantitative evaluation should have been included with respect to accuracy and other measures.

6 CONCLUSION

This report has introduced our device that allows users to press a button, ask a question, and receive an answer to their question without relying on others for help. We discussed how this has the potential to help the increasingly high number of people that have visual impairments, to give them more independence in their lives by helping them to understand their surroundings. The implementation of the device in hardware and software was introduced.

Conclusion is not effective.
Conclusion does not state the main findings (good and bad both).

Future work should be a short text. One paragraph of 250 to 300 words only.

Ironically, conclusion (the work of group) is shorter than future work.

7 FUTURE WORK

Due to limitations we could not include some of the features within the scope of the project. The primary area of improvement that could be made to the device is to increase its specificity to specific tasks. While the state-of-the-art machine learning models for visual question answering are very effective at giving a general impression of a user's surroundings, it is hard to trust their output. This is due to them often getting small details wrong, which leads to them potentially misleading the users of the device. Making the device more specific to a use-case may have the opportunity to optimise the device for those use-cases, which could lead to better results. ✓

Additionally, by integrating the device with GPS and online maps, we could answer spatial questions more effectively using the device. This could include the detection of landmarks for helping visually impaired people to navigate their environment. This could help to further improve the autonomy of users of this device. ✓

REFERENCES

[1] Tejal Adep, Rutuja Nikam, Sayali Wanewe, and Ketaki Naik. 2021. Visual Assistant for Blind People using Raspberry Pi. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* (06 2021), 671–675. https://doi.org/10.32628/CSEIT2173142 ✓

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). arXiv:2005.14165 https://arxiv.org/abs/2005.14165

[3] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. 2019. Visual Wake Words Dataset. arXiv:1906.05721 http://arxiv.org/abs/1906.05721

[4] Tedros Ghebreyesus. 2019. https://www.who.int/publications/i/item/9789241516570

[5] Google. 2020. Google Lookout: App reads grocery labels for blind people. https://www.bbc.com/news/technology-53753708

[6] Hyun K. Kim, Sung H. Han, Jaehyun Park, and Joohwan Park. 2016. The interaction experiences of visually impaired people with assistive technology: A case study of smartphones. *International journal of industrial ergonomics* 55 (2016), 22–33.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[8] Wai-Ying Low, Mengqiu Cao, Jonas De Vos, and Robin Hickman. 2020. The journey experience of visually impaired people on public transport in London. *Transport policy* 97 (2020), 137–148.

[9] Fernando Pérez, Brian E. Granger, and John D. Hunter. 2011. Python: An Ecosystem for Scientific Computing. *Computing in Science & Engineering* 13, 2 (2011), 13–21. https://doi.org/10.1109/MCSE.2010.119

[10] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust Speech Recognition via Large-Scale Weak Supervision. https://cdn.openai.com/papers/whisper.pdf ✓

[11] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. 2022. OFA: Unifying Architectures, Tasks, and Modalities Through a Simple Sequence-to-Sequence Learning Framework.

[12] Tess Winlock, Eric Christiansen, and Serge Belongie. 2010. Toward real-time grocery detection for the visually impaired. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*. 49–56. https://doi.org/10.1109/CVPRW.2010.5543576

8 APPENDIX

The *audio.py*, *program.py*, and *webserver.py* source code files are given here, as appendices. The code of webservers to host the OFA models was also modified to account for the use-case of this project. However, their code is not included here, as only small modifications were made, and they are very large programs.

8.1 audio.py

The *audio.py* file contains utility functions to deal with the audio recording and the notification sound playing of the device.

```
import pyaudio
import wave
import os
import time
import math
import numpy as np
from pydub import AudioSegment
from pydub.playback import play
import librosa


# Audio recording settings.
form_1 = pyaudio.paInt16
chans = 1
samp_rate = 44100
chunk = 4096
silent_secs = 2
max_duration = 15
silent_threshold = 500
dev_index = 1

# The sound used for notifications to the user.
notification_sound = AudioSegment.from_mp3("ping.mp3")


def lower_pitch(sound):
    """ Lowers the pitch of the given sound. """
    n_steps = -1
    y = np.frombuffer(notification_sound._data, dtype=np.int16).astype(np.float32)/(2**15)
    y = librosa.effects.pitch_shift(y, sr=notification_sound.frame_rate, n_steps=n_steps)
    y = np.array(y * (1 << 15), dtype=np.int16)[::2].tobytes()
    return AudioSegment(y, frame_rate=notification_sound.frame_rate, sample_width=2, channels=1)


# Create a lower pitch version of the notification sound
# for use when ending recording or finding.
lower_pitch_notification_sound = lower_pitch(notification_sound)
```

✓ ✓

```
def list_audio_devices():
    """
    Lists all available audio devices, and their indices.
    The dev_index configuration parameter should be set
    to the index of the desired output.
    """
    for ii in range(audio.get_device_count()):
        print(ii, audio.get_device_info_by_index(ii).get('name'))


def play_notification_sound(low_pitch = False):
    """ Plays a notification sound. """
    if not low_pitch:
        play(notification_sound)
    else:
        play(lower_pitch_notification_sound)



def record_prompt():
    """ Records an audio prompt from the user. """
    try:
        # Opens a stream to record the audio from
        # the microphone.
        audio = pyaudio.PyAudio()
        stream = audio.open(
            format = form_1,
            rate=samp_rate,
            channels=chans,
            input_device_index = dev_index,
            input=True,
            frames_per_buffer=chunk
        )

        # Play a notification sound to indicate the
        # start of recording.
        print("recording")
        play_notification_sound()
        audio_data = bytes()

        # Record the audio by chunks, and keep track of the
        # number of total and silent frames that have been
        # recorded.
        silent_frames = 0
        total_frames = 0

        # Continue recording frames until hitting the maximum
        # number of silent seconds, or the maximum number of
        # total seconds.
```

✓

```
        while silent_frames < (samp_rate / chunk) * silent_secs \
                and total_frames < (samp_rate / chunk) * max_duration:

            # Read a single chunk of audio.
            data = stream.read(chunk, exception_on_overflow = False)
            audio_data += data
            total_frames += 1

            # Use the root-mean squared value to measure volume.
            rms = math.sqrt((np.frombuffer(data, np.int16).astype(float)**2).mean())
            if rms < silent_threshold:
                silent_frames += 1
            else:
                silent_frames //= 2

        # Play a notification sound to indicate the end of recording.
        play_notification_sound(True)
        print("finished recording")

        # Resample audio to 16kHz.
        resampling_factor = samp_rate / 16000
        audio_data = np.frombuffer(audio_data, np.int16).astype(np.float32)
        audio_data = np.interp(
            np.arange(0, len(audio_data), resampling_factor),
            np.arange(0, len(audio_data)),
            audio_data
        )
        return audio_data.astype(np.int16).tobytes()

    finally:
        # Cleanup the audio streams.
        stream.stop_stream()
        stream.close()
        audio.terminate()
```

✓

8.2 program.py

The *program.py* file contains the main program loop, and various functions to perform the steps needed for the device to function.

```
"""
The main program that is run on the Raspberry Pi.
"""
import time
import requests
import base64
import RPi.GPIO as GPIO
import time
from picamera2 import Picamera2
import pyttsx3
import audio
from audio import play_notification_sound
import cv2
import numpy as np


# The IP address of the server that is running the machine learning models.
server_ip = "172.20.10.2"

# The location to save the photos to before they are sent.
img_file = "photo.jpg"
img_w = 406
img_h = 304

# The GPIO pin used for the button.
button_pin = 21

# Initialise the GPIO pin for the button.
GPIO.setmode(GPIO.BCM)
GPIO.setup(button_pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)


def speak(text):
    """ Plays the given text as audio through the earphones. """
    print(f"Speak: {text}")
    engine = pyttsx3.init()
    engine.setProperty('rate', 150)
    engine.setProperty('voice', "english_rp")
    engine.say(text)
    engine.runAndWait()


def take_photo(camera):
    """ Capture a photo from the camera. """
    raw_photo = camera.capture_array()
```

✓

✓

```
    # Downscale the photo as it will be huge initially.
    photo = cv2.resize(raw_photo, (img_w, img_h), interpolation=cv2.INTER_AREA)[::-1,::-1,::-1]
    return photo


def read_photo_data(img_file):
    """ Reads the JPG bytes of the given image. """
    with open(img_file, "rb") as f:
        return f.read()


def pack_request_data(audio_data, img_file):
    """
    Packs the audio recording and image into a single
    bytes object to send to the server.
    """
    photo_data = read_photo_data(img_file)

    lengths = np.array([len(audio_data), len(photo_data)], dtype=np.int32)
    lengths_data = lengths.tobytes()
    assert(len(lengths_data) == 8)

    return lengths_data + audio_data + photo_data


def describe_object(data):
    """
    Uses the detected location of an object
    to describe its location through audio.
    """
    prompt = data["prompt"]
    x1, y1, x2, y2 = data["location"]
    size_pixels = (x2 - x1) * (y2 - y1)
    central_x = (x1 + x2) // 2
    central_y = (y1 + y2) // 2

    if size_pixels < (img_w // 10)**2:
        size = "tiny"
    elif size_pixels < (img_w // 5)**2:
        size = "small"
    elif size_pixels < (img_w // 2)**2:
        size = "moderately sized"
    else:
        size = "large"

    third_w = img_w / 3
    third_h = img_h / 3
    if central_x <= third_w:
        if central_y <= third_h:
```

✓

```
            position = "in the top left"
        elif central_y <= 2 * third_h:
            position = "to the left"
        else:
            position = "in the bottom left"
    elif central_x <= 2 * third_w:
        if central_y <= third_h:
            position = "towards the top"
        elif central_y <= 2 * third_h:
            position = "central"
        else:
            position = "towards the bottom"
    else:
        if central_y <= third_h:
            position = "in the top right"
        elif central_y <= 2 * third_h:
            position = "to the right"
        else:
            position = "in the bottom right"

    return f"{prompt} is {size} and {position}"


def is_button_pressed():
    """ Detects whether the button is currently pressed. """
    return GPIO.input(button_pin)
```

✓

```
# Run the program.
if __name__ == "__main__":

    # Setup the camera.
    with Picamera2() as camera:
        camera.configure(camera.create_still_configuration())
        camera.start()

        # Wait until the button is pressed.
        while True:
            print("Waiting for button press...")
            while not is_button_pressed():
                time.sleep(0.1)
            print("Button pressed")

            # Record the audio prompt and take a photo.
            audio_data = audio.record_prompt()
            photo = take_photo(camera)
            cv2.imwrite(img_file, photo)

            # Send the audio and photo to the server for
            # processing.
            result = requests.post(
                url=f"http://{server_ip}:9888/analyse",
                data=pack_request_data(audio_data, img_file)
            )
            json = result.json()

            if "text" in json:
                # Received a simple text response.
                # Speak it out for the user.
                speak(json["text"])
            else:
                # Received a find-mode response.
                # Speak it out for the user, and
                # capture another image to look
                # for the object.
                obj_desc = json["object"]
                speak(describe_object(obj_desc))
                prompt = obj_desc["prompt"]

                # Repeat taking photos to find the object
                # until the button is pressed.
                while not is_button_pressed():
                    # Wait for 2 seconds for the user to move.
                    for i in range(20):
                        time.sleep(0.1)
                        if is_button_pressed():
                            break

                    # Take another photo.
                    photo = take_photo(camera)
                    cv2.imwrite(img_file, photo)

                    # Send the photo to the server.
                    photo_data = read_photo_data(img_file)
                    result = requests.post(
                        url=f"http://{server_ip}:9888/find",
                        data=photo_data,
                        params={"prompt": prompt}
                    )
                    json = result.json()

                    if "text" in json:
                        # If find-mode has stopped, the server
                        # will send a text message to the user.
                        speak(json["text"])
                        break
                    else:
                        # Describe the location of the object.
                        speak(describe_object(json["object"]))
```

✓

```
                # Play a notification sound to indicate that
                # object finding has stopped.
                play_notification_sound(True)
```

✓

8.3 webserver.py

The *webserver.py* file contains the API endpoints that the device sends its photos and prompts to. The API responds with the text to say to the user, and potentially the location of an object if a find command is issued by the user.

✓

```python
import json
import traceback

import flask
import numpy as np
from flask import request
import whisper
import time
import base64
import requests
from string import punctuation, whitespace


# Create a Flask webserver.
app = flask.Flask(__name__)

# Load the Whisper model for audio transcription.
whisper_model = whisper.load_model("base.en")

# The local URLs to the hosted grounding and visual
# question answering machine learning models.
grounding_url = "http://127.0.0.1:7860"
vqa_url = "http://127.0.0.1:7861"


def send_gradio_request(url, jpeg_data, prompt) -> str:
    """ Sends a request to one of the machine learning models hosted using Gradio. """
    start = time.time()
    # Encode the image into base64 to send to the model.
    img_base64 = "data:image/jpeg;base64," + base64.b64encode(jpeg_data).decode("utf-8")

    # Send the image and prompt for prediction.
    r = requests.post(
        url=f"{url}/api/predict/",
        json={"data": [img_base64, prompt]}
    )
    result = r.json()["data"][0]
    print(f"* Response took {time.time() - start:.2f} seconds")
    return result


def ground(jpeg_data, prompt) -> dict:
    """
    Sends the photo and prompt to the
    grounding model for object detection.
    """
    print(f"Grounding \"{prompt}\"")
    json_response = send_gradio_request(grounding_url, jpeg_data, prompt)
    return {
        "prompt": prompt.strip(punctuation + whitespace).lower(),
        "location": json.loads(json_response)
    }


def answer(jpeg_data, question) -> str:
    """
    Sends the photo and question to the
    visual question answering model.
    """
    print(f"Answering \"{question}\"")
    return send_gradio_request(vqa_url, jpeg_data, question)


def transcribe(audio: bytes) -> str:
    """
    Transcribes the raw audio data into text using Whisper.
    """
    # Convert to a NumPy array and scale for Whisper.
    audio = np.frombuffer(audio, np.int16).astype(np.float32) / 32768.0

    # Detect text using Whisper.
    audio = whisper.pad_or_trim(audio)
    mel = whisper.log_mel_spectrogram(audio).to(whisper_model.device)
    options = whisper.DecodingOptions(language="en", fp16=False)
    result = whisper.decode(whisper_model, mel, options)
    return result.text


@app.route('/analyse', methods=['POST'])
def analyse():
    """
    The main webserver entrypoint that transcribes the
    audio data and performs the user's request.
    """
    try:
        # Read the audio and photo from the request.
        data = request.get_data()
        lengths = np.frombuffer(data[0:8], np.int32)
        audio_data = data[8:8 + lengths[0]]
        photo_data = data[8 + lengths[0]:]

        # Transcribe the audio data to text.
        question = transcribe(audio_data)
        # Detect common types of prompts.
        stripped_question = question.strip(whitespace + punctuation).lower()
        if stripped_question == "":
            # No prompt text was detected.
            return {"text": "Sorry, I could not hear you"}
        elif stripped_question == "describe":
            # Shorthand to ask for a description of the image.
            return {"text": answer(photo_data, " what does the image describe?")}
        elif stripped_question.startswith("find "):
            # Asks for the model to find an object in the image.
            index = question.lower().index("find ")
            prompt = question if index < 0 else question[index + len("find "):]
            prompt = prompt.strip(whitespace + punctuation).lower()

            # Check if the object can be detected.
            contains_object = answer(photo_data, f"is there a {prompt} in the image?")
            if contains_object.strip(whitespace + punctuation).lower() == "no":
                return {"text": f"{prompt} could not be found"}

            # Detect the location of the object.
            return {"object": ground(photo_data, prompt)}
        else:
            # Standard visual question answering.
            return {"text": answer(photo_data, question)}

    except:
        # In the unfortunate case of an error, report it to the user.
        traceback.print_exc()
        return {
            "text": "there was an error"
        }


@app.route('/find', methods=['POST'])
def find():
    """
    The endpoint for receiving new photos for finding
    an object. In this endpoint, no audio transcription
    needs to be performed.
    """
    try:
        # Get the photo and prompt from the request.
        photo_data = request.get_data()
        prompt = request.args.get("prompt")

        # Check whether the photo contains the object.
        contains_object = answer(photo_data, f"is there a {prompt} in the image?")
        print(contains_object)
        if contains_object.strip(whitespace + punctuation).lower() == "no":
            return {"text": f"{prompt} could not be found"}
        # Detect the location of the object.
        return {"object": ground(photo_data, prompt)}

    except:
        # In the unfortunate case of an error, report it to the user.
        traceback.print_exc()
        return {
            "text": "there was an error"
        }


# Run the webserver and make it available to all devices on the same network.
app.run(host="0.0.0.0", port=9888)
```

✓