



Robotic AI Bear using ChatGPT

Created by Melissa LeBlanc-Williams



<https://learn.adafruit.com/robotic-ai-bear-using-chatgpt>

Last updated on 2024-03-08 04:13:04 PM EST

Table of Contents

Overview	5
<ul style="list-style-type: none">• Parts• Easier Build Option• Compact Build Option• Optional Parts	
Circuit Diagram	8
<ul style="list-style-type: none">• Required Wiring• Compact Build Additional Wiring	
3D Printing	9
<ul style="list-style-type: none">• Slicer Settings• Parts Details	
Modifying the Bear	12
<ul style="list-style-type: none">• Disassembly• Preparing the Motor Wires• Preparing the Foot Button• Preparing the Speaker	
DC and Stepper Motor HAT Assembly	20
<ul style="list-style-type: none">• Solder on Headers and Terminal Block• And Solder!	
Wiring	27
<ul style="list-style-type: none">• USB Cable Modification Option• USB Cable Adapter Option• Foot Button Connection• MAX98357 I2S Amplifier	
Raspberry Pi Setup	33
<ul style="list-style-type: none">• Install Raspberry Pi OS 64-bit Lite• Audio System and Drivers• I2S Audio Setup for Compact Build Option• USB Audio Selection for Easier Build Option• Install Required Libraries	
Create an Account with OpenAI	36
Create an Account with Azure	38
<ul style="list-style-type: none">• Using a Different Voice	
Code the Bear	42
<ul style="list-style-type: none">• Downloading the Code• How the Python Code Works	
Assembly	52
<ul style="list-style-type: none">• Assembly Tool• Case Bottom• Case Top• Final Case Assembly	

- [USB Speaker](#)
- [USB Microphone](#)
- [Power Connections](#)
- [Final Bear Assembly](#)

Usage

66

- [Troubleshooting](#)
-

Overview

The [Peek-A-Boo Teddy Bear by GUND](https://adafru.it/18BQ) (<https://adafru.it/18BQ>) is an affordable robotic bear that plays pre-recorded sounds. It has a motorized mouth to simulate talking and motorized arms that can lift a blanket to hide its face.

This guide will show you how to enhance the bear to make it do more than just play 'peek-a-boo' by tying it to OpenAI's ChatGPT API. Now you can speak to the toy and ask it questions, have it tell you stories, and more! Voice recognition is used to generate the prompts and then text-to-speech plays back the response

We do this by replacing the electronics inside with a [Raspberry Pi 4](http://adafru.it/4292) (<http://adafru.it/4292>), an [Adafruit Motor HAT](http://adafru.it/2348) (<http://adafru.it/2348>), and a couple of options for playing audio such as the [MAX98357 I2S Amp](http://adafru.it/3006) (<http://adafru.it/3006>) and a [Mini USB Speaker](http://adafru.it/3369) (<http://adafru.it/3369>). The Raspberry Pi 4 will be running a custom Python script that interacts with multiple APIs including two from OpenAI and one from Azure.

It takes inspiration from some iconic bear characters including a popular toy in the 1980s called [Teddy Ruxpin](https://adafru.it/18BS) (<https://adafru.it/18BS>) that would tell stories while making prerecorded robotic gestures and movements. This project was also inspired by the toy bear from the movie [A.I.](https://adafru.it/18BU) (<https://adafru.it/18BU>), which was advanced robotic companion designed to provide love and comfort to a child.

This project combines the best of both worlds to create a robotic bear that not only tells stories, jokes, and poems, but also interacts with you using natural language processing powered by ChatGPT. This technology allows the bear to respond to questions and engage in conversations.

By the end of this guide, you will have learned how to upgrade your own robotic bear with ChatGPT integration, gaining hands-on experience with robotics, programming, and artificial intelligence.

Parts

This project has a few options depending on how compact versus how easy you would like the project. Here are the parts you will need for either build.

The first thing you will need is a Raspberry Pi 4. This project will likely also work on a Raspberry Pi 3 if you skip the custom enclosure, but it has not been tested.

You may also be able to use another board, but it likely won't run Raspberry Pi OS, so the software setup will likely be significantly different and covering other boards is beyond the scope of this guide.

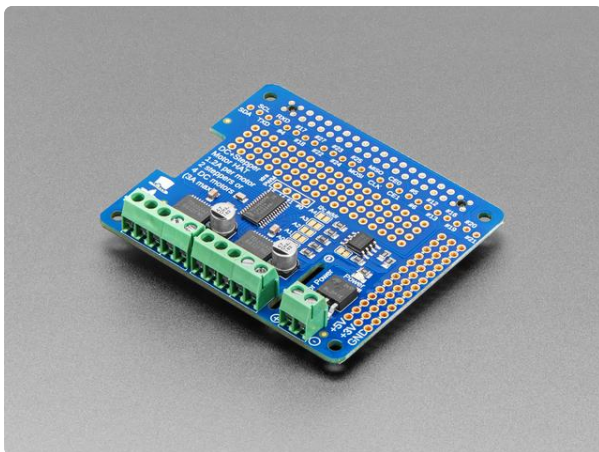


Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B is the newest Raspberry Pi computer made, and the Pi Foundation knows you can always make a good thing better! And what could make the Pi 4 better...

<https://www.adafruit.com/product/4297>

You will also need a motor HAT. The reason for the HAT over the bonnet is because more soldering points are available.



Adafruit DC & Stepper Motor HAT for Raspberry Pi - Mini Kit

Let your robotic dreams come true with the new DC+Stepper Motor HAT from Adafruit. This Raspberry Pi add-on is perfect for any motion project as it can drive up to 4 DC or 2 Stepper...

<https://www.adafruit.com/product/2348>

You will need this set of nylon screws and standoffs for the project.



Black Nylon Machine Screw and Stand-off Set – M2.5 Thread

Totaling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You'll have enough screws, nuts, and hex standoffs to fuel your maker...

<https://www.adafruit.com/product/3299>

The bear that you will be modifying

1 x [GUND Peek-A-Boo Teddy Bear](#)

<https://amzn.to/3IUwh7>

The bear that you will be modifying

1 x [Mini USB Microphone](#)

<https://www.adafruit.com/product/3367>

You can use this or any USB microphone

1 x [USB to 2.1mm Male Barrel Jack Cable](#)

<https://www.adafruit.com/product/2697>

Cheap USB plug to wire option

1 x [Premium Male/Male Jumper Wires](#)

<https://www.adafruit.com/product/1957>

6" x 20 wires

1 x [3 Amp Battery Bank](#)

<https://amzn.to/3nryAHX>

10000mAh 5V 3A Battery Pack Charger

Easier Build Option

1 x [Mini External USB Stereo Speaker](#)

<https://www.adafruit.com/product/3369>

USB-powered Speaker

1 x [Female DC Power adapter](#)

<https://www.adafruit.com/product/368>

For easily connecting USB power

Compact Build Option

1 x [Adafruit I2S 3W Class D Amplifier Breakout](#)

<https://www.adafruit.com/product/3006>

MAX98357 Amplifier

1 x [Mini Metal Speaker](#)

<https://www.adafruit.com/product/1890>

Optional 8 ohm 0.5W speaker upgrade

1 x [100K Through-Hole Resistors](#)

<https://www.adafruit.com/product/2787>

For reducing the gain on the amp

1 x [USB Type A to Type C Cable](#)

<https://www.adafruit.com/product/4473>

Short USB Cable

Optional Parts

These parts are optional, but enhance the project.

Heat shrink tubing to prevent wires from shorting

1 x Heat Shrink Pack

<https://www.adafruit.com/product/344>

Heat shrink tubing to prevent wires from shorting

1 x Aluminum Heat Sink for Raspberry Pi

<https://www.adafruit.com/product/3083>

This fits perfectly under the motor HAT

1 x 2-pin JST SM Plug + Receptacle Cable Set

<https://www.adafruit.com/product/2880>

For easily disconnecting the foot button from the pi

1 x Wire Ferrule Kit

<https://www.adafruit.com/product/5131>

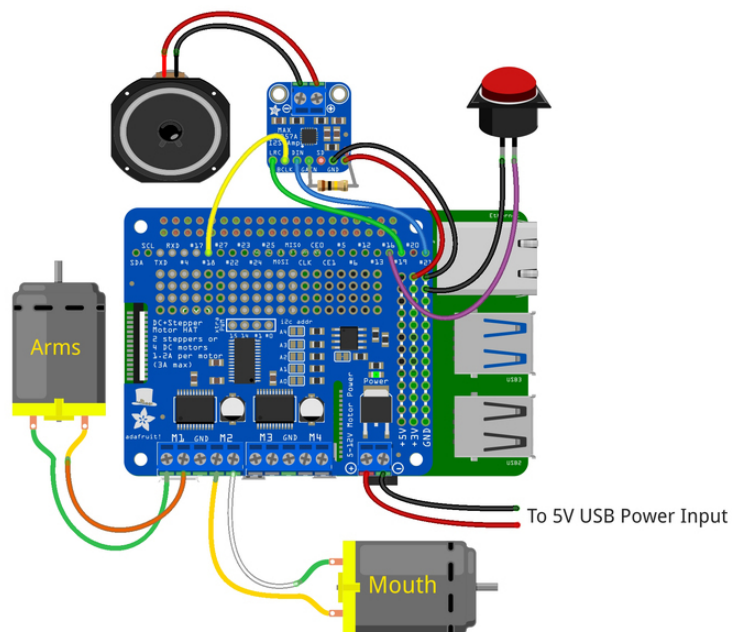
This helps in certain situations, but most of the wiring is too small gauge

1 x USB A Extension Cable

<https://www.adafruit.com/product/4055>

If you'd like to position the USB microphone better

Circuit Diagram



fritzing

Required Wiring

The HAT Motor Power will come from a USB cable that will be plugged into the battery pack.

- **Green Arms Motor Wire to HAT M1 left side (1st terminal position)**
- **Orange Arms Motor Wire to HAT M1 right side (2nd terminal position)**

- **Yellow** Mouth Motor Wire to **HAT M2 left side (4th terminal position)**
- **White** Mouth Motor Wire to **HAT M2 right side (5th terminal position)**
- Foot Button wire 1 to **HAT GPIO #16**
- Foot Button wire 2 to **HAT Gnd**
- **HAT Motor Power +** to External USB +5V
- **HAT Motor Power -** to External USB Gnd

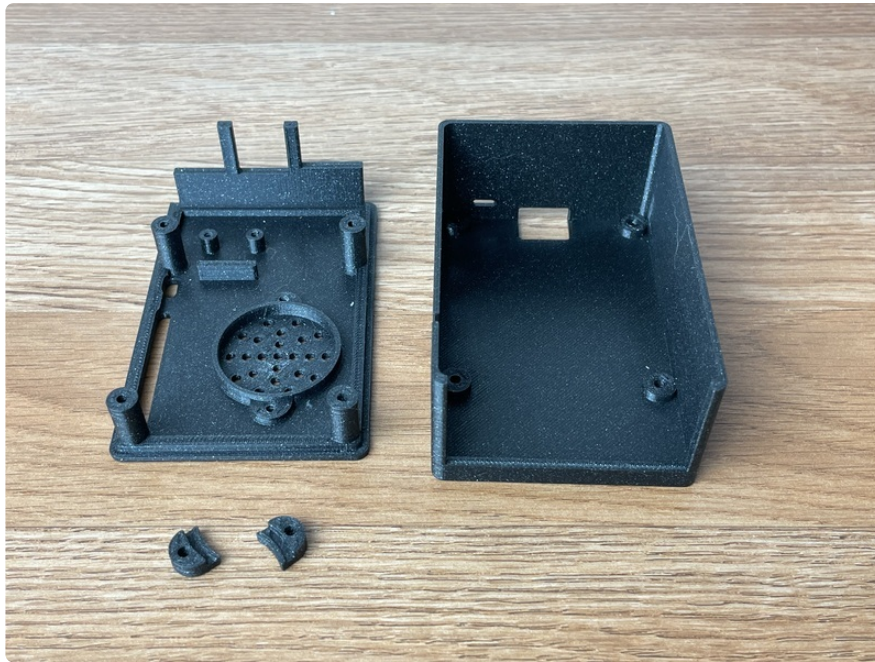
Compact Build Additional Wiring

For the speaker wires, the order doesn't matter. Since it only has a single speaker.

- **MAX98357 Vin** to **HAT 3V**
- **MAX98357 Gnd** to **HAT Gnd**
- **MAX98357 LRC** to **HAT GPIO #19**
- **MAX98357 BCLK** to **HAT GPIO #18**
- **MAX98357 DIN** to **HAT GPIO #21**
- 100K Resistor to **MAX98357 Gain**
- Other side of 100K Resistor to **MAX98357 Vin**
- Speaker Wires to **MAX98357 Speaker Output**

3D Printing

If you are using another board for this build, the case likely won't fit. You can either modify the design for your needs or go without the case.



This project will be assembled with just a few 3D printed parts, described below. The parts are available in either the STL (older) or 3MF (newer) format. Most slicers support both. You'll need to print multiples of some parts:

Main Case

- 1 x **Case Top.stl**
- 1 x **Case Bottom.stl**

Speaker Retainers

If you're using the 4mm 0.25W speaker from inside the bear:

- 2 x **4mm Thick Speaker Retainer.stl**

If you're using an upgraded Adafruit 2mm 0.5W speaker:

- 2 x **2mm Thick Speaker Retainer.stl**

Download STL Files

<https://adafru.it/19bN>

Download 3MF Files

<https://adafru.it/18BW>

Download CAD Source

<https://adafru.it/18BY>

Slicer Settings

The specific setting values aren't that important with this case. However, here's the settings used to print the case for this guide.

For the larger case parts:

- 20% infill
- Supports on build plate enabled
- 0.2mm Layer Height

For the small speaker retainers:

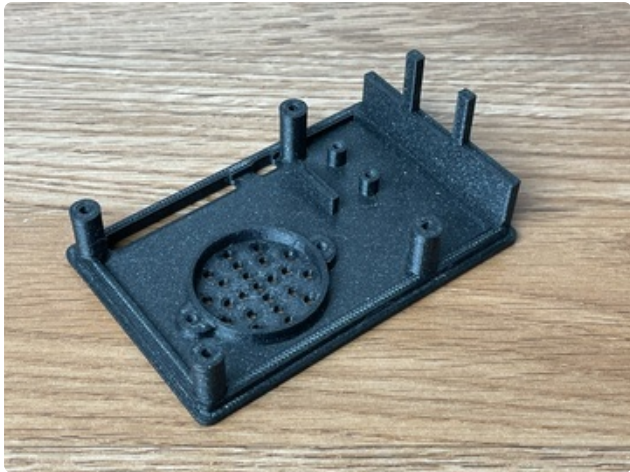
- 20% infill
- 0.2mm Layer Height
- No Supports

Parts Details



Case Bottom

The Raspberry Pi and Motor HAT will be housed here



Case Top

The speaker and MAX98357 will be attached here



4mm Thick Speaker Retainer

These retainers are for the 4mm thick speaker that you can harvest from inside the bear.



2mm Thick Speaker Retainer

These are for holding the thinner 2mm 0.5W Adafruit speaker upgrade.

Modifying the Bear

To be able to make use of the bear, you will need to disassemble it and remove the circuitry so that it can be connected to the Raspberry Pi. Some of the tools you will need include a couple of sizes of phillips-head and Flathead screwdrivers, some isopropyl alcohol, and flush cutters or a soldering iron.

Some screws will be reused. Be sure to keep those in a place where you can find them later.

Disassembly

As you disassemble the bear, be sure to carefully set aside the screws and electronics as some of this will be re-used in the final circuit.

Isopropyl Alcohol is very good at removing hot glue.



Start by removing the bear from the packaging.



Turn the bear over and open the flap covering the battery compartment. You may want to remove the battery cover and batteries so you can use them for something else. They won't be used in this project.



Remove the 4 screws and set them aside. Then remove the cover. You may need to pry it a bit.



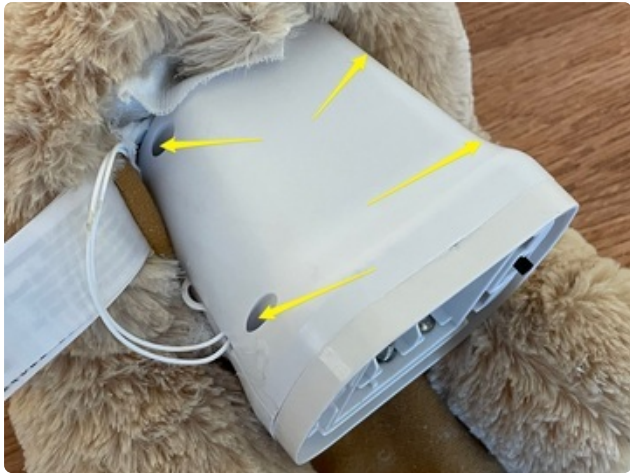
The fabric around the electronics housing should now be free to pull away. Go ahead and pull the housing out of the bear.



To expose the housing, first carefully push the cloth around the housing up so that the bottom half inch of the housing is exposed. Then push it up the rest of the way.



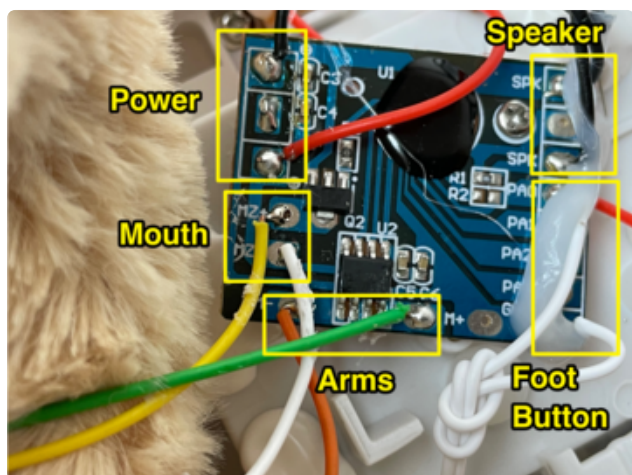
The housing is attached further up with a bunch of hot glue. Applying some isopropyl alcohol (or IPA) to the glue with a cotton swab or something similar will cause the glue to become brittle and less adhesive. The higher the concentration of IPA, the better it works. Don't forget to remove the glue on the lower white wires.



There are 4 more screws holding the housing together. Remove those screws and set them aside.



Once the two halves of the housing are taken apart, you will have access to the electronics and wiring.



On the circuit board are connections to the power, mouth motor, arms motor, foot button, and speaker. Remove everything except the power wires because they won't be needed. If you are doing the easier build option or upgrading the speaker, you won't need the built-in speaker either. You can either use flush cutters or remove the glue and desolder each wire.

Preparing the Motor Wires

In order to connect the motor wires to the Adafruit Motor HAT, we recommend extending the thin wires with thicker jumper wires and a nice connector on the end.

If you don't mind the wires being short on the short side, you may want to try wire ferrules. These have the advantage of not being as deep and you don't risk melting the polyfill stuffing inside the bear. However, the wire gauge is a bit small and it's tricky to get the ferrules to stay on properly.

You could also combine the two methods and extend the wires with thicker ones and replace the long black connectors with the shorter ferrules.

Be very careful while using heat near the bear! It's easy to accidentally melt the fur or the polyfill stuffing! If you aren't very confident with your skills, you could temporarily line the fur with aluminum foil to act as a heat shield.



With the 6" wires, by cutting in half, you can have 2 useable parts. If you would like longer wires, you can just cut off 1 end.



You'll want to cut, strip, and tin 4 wires to match the motor wires. You will also want to cut 4 pieces of heat shrink tubing. Place the heat shrink on the wires now so you don't forget later.



Strip and tin the wires inside the bear. Very carefully solder the wire extensions to the motor wires sticking out. You may want to temporarily remove some of the stuffing to give you more room to work and place some aluminum foil around the area you are working.

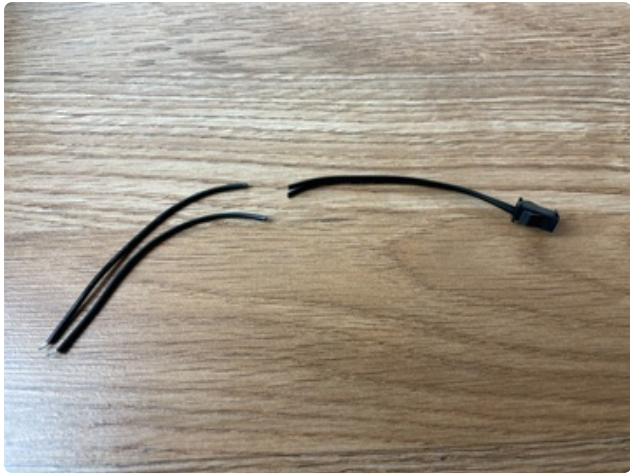


Now slide the heat shrink tubing up and heat it. I used a hot air soldering station set at 120° Celsius, which seemed to be just hot enough to shrink it. You could also use electrical tape to insulate if you aren't comfortable with heat shrink.

Preparing the Foot Button

If you would like to make it so the foot button can be disconnected, you can attach the JST SM Plug cable to it and the receptacle to the Pi.

If you are not using this connector, just strip and tin the pair of white wires coming out of the bear with a small knot in the end without shortening the wires.



The foot button wires and JST cables are quite long, so it's helpful to trim them down a bit.



Place some heat shrink tubing on the foot button wires. Since the wires are already separated, it make things a lot easier.



Strip, tin, and solder the harness to the foot button wires. The specific wire doesn't matter since pushing the button connects them together.



Slide the heat shrink tubing over the soldered wires and heat it up. Since the wiring isn't as short as the motor wires, this is much easier.



That's it. You'll connect the receptacle on the Wiring page.

Preparing the Speaker

If you are reusing the speaker and you would like to use the terminal connector on the MAX98357, you will want to replace the wires on the speaker with ones similar to the motor. Otherwise, you can just strip and tin the existing wires.



Prepare two more wires and desolder the ones that come with the speaker.



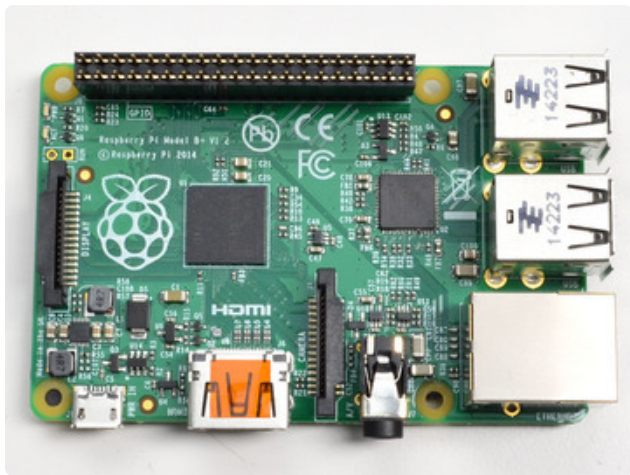
Solder the new wires onto the speaker.

That's it! The bear is all prepared for adding the new electronics.

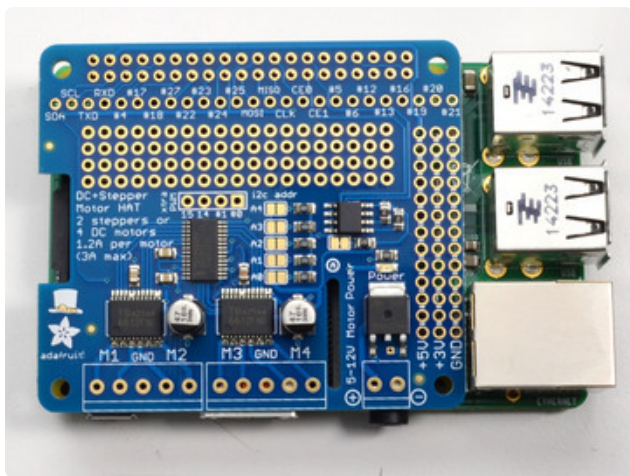
DC and Stepper Motor HAT Assembly

Solder on Headers and Terminal Block

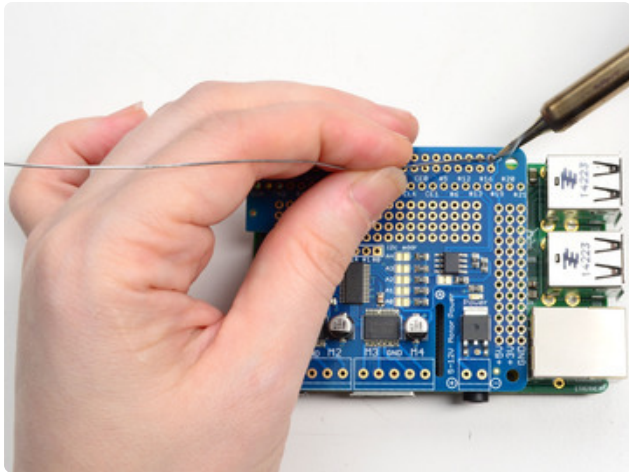
Before we can motorin' there's a little soldering to be done. This step will attach the 2x20 socket header so that we can plug this HAT into a Raspberry Pi, and the terminal blocks so you can attach external power and motors.



Start by plugging the 2x20 header into a Raspberry Pi, this will keep the header stable while you solder. Make sure the Pi is powered down!



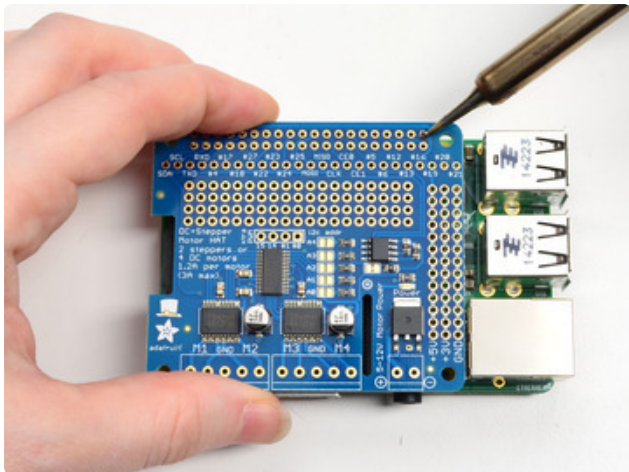
Place the HAT on top so that the short pins of the 2x20 header line up with the pads on the HAT



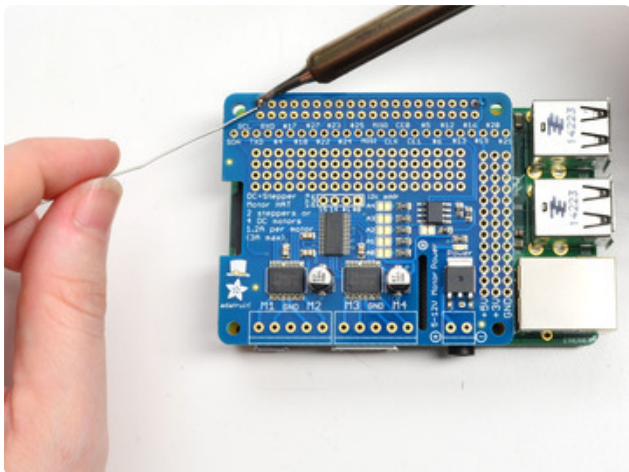
And Solder!

Heat up your iron and solder in one header connection on the right.

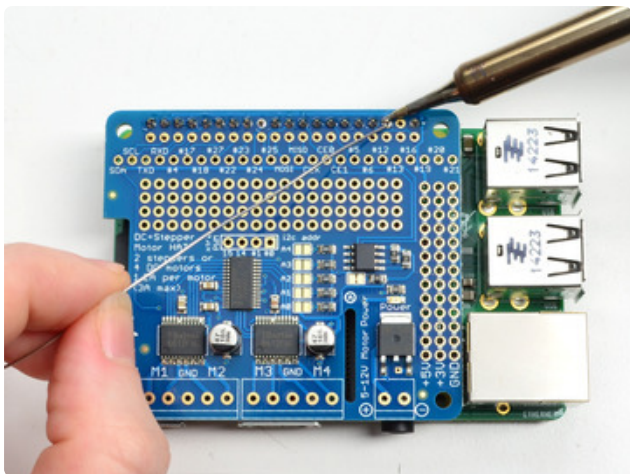
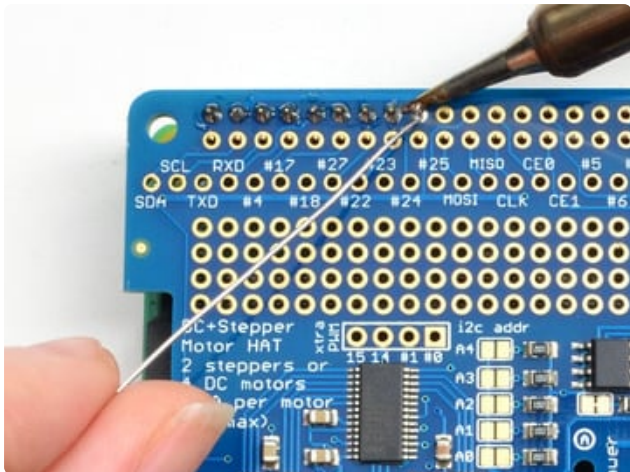
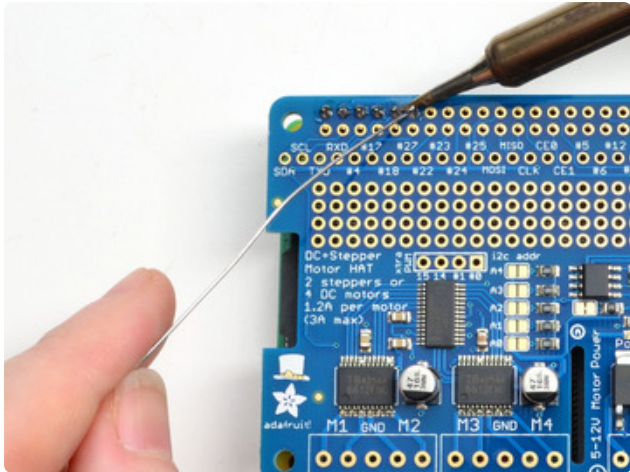
Once it is soldered, put down the solder and reheate the solder point with your iron while straightening the HAT so it isn't leaning down



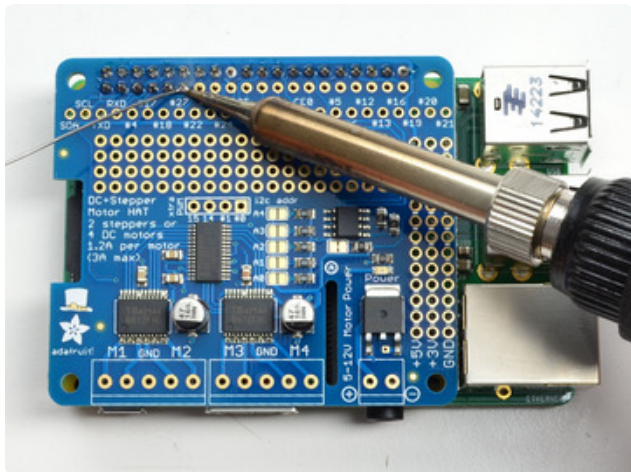
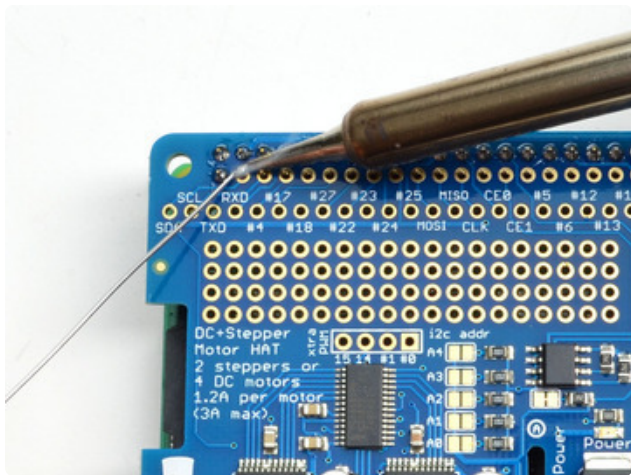
(For tips on soldering, be sure to check out our [Guide to Excellent Soldering \(https://adafruit.it/aTk\)](https://adafruit.it/aTk)).



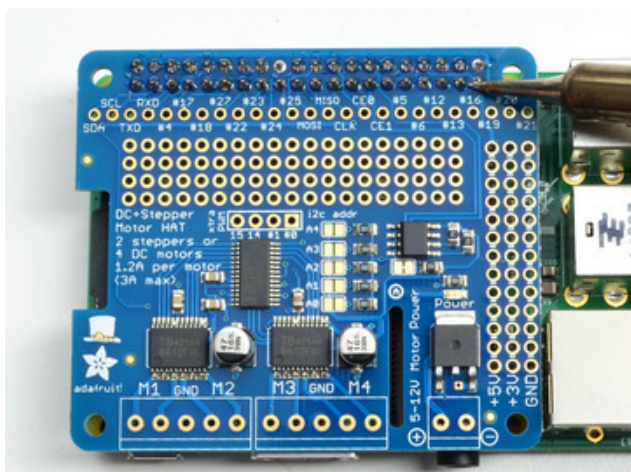
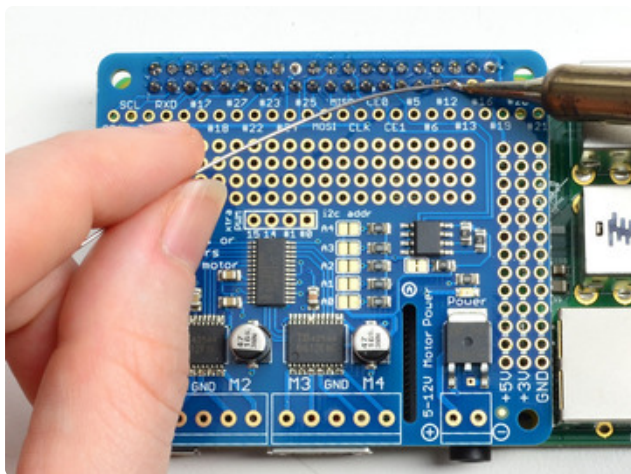
Solder one point on the opposite side of the connector

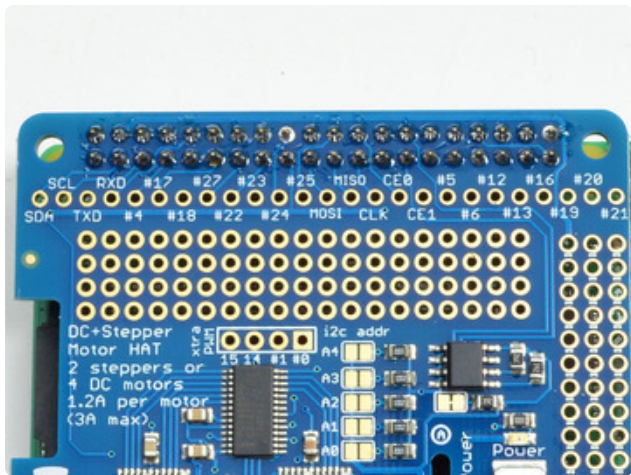


Solder each of the connections for the top row

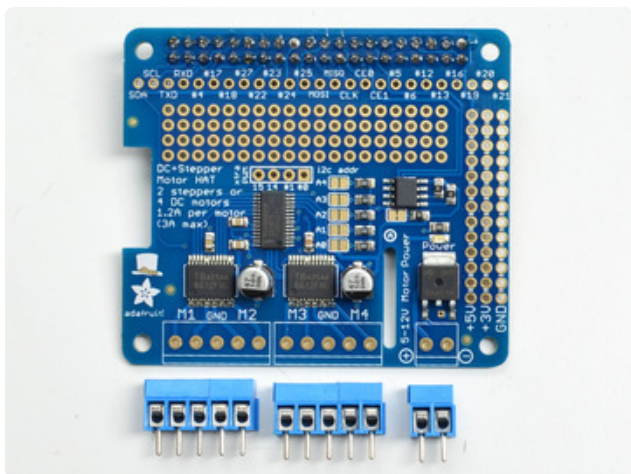


Flip the board around and solder all the connections for the other half of the 2x20 header





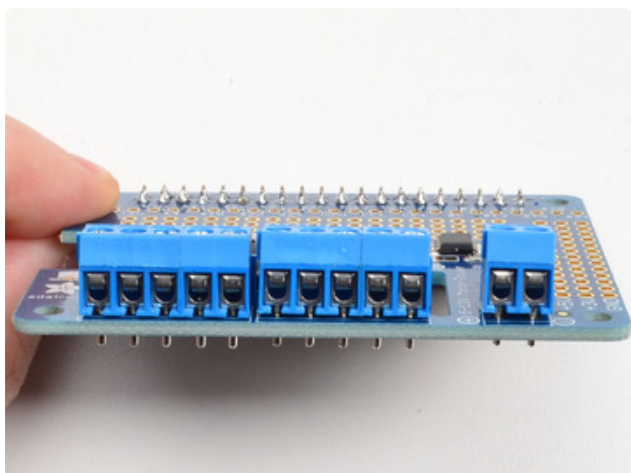
Check over your work so far, make sure each solder point is shiny, and isn't bridged or dull or cracked



Now grab the 3.5mm-spaced terminal blocks. These will let you quickly connect up your motor and power supply using only a screw driver

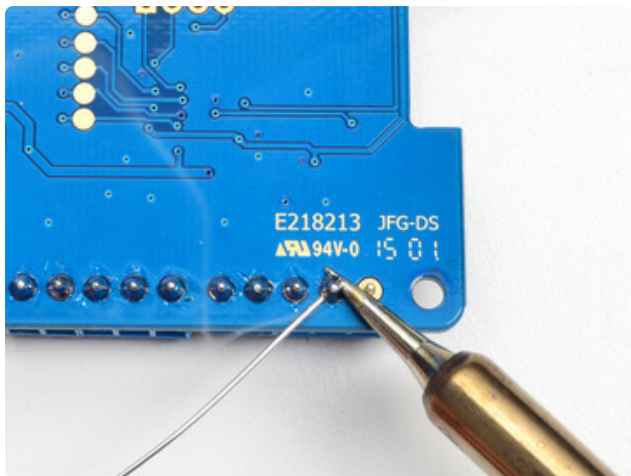
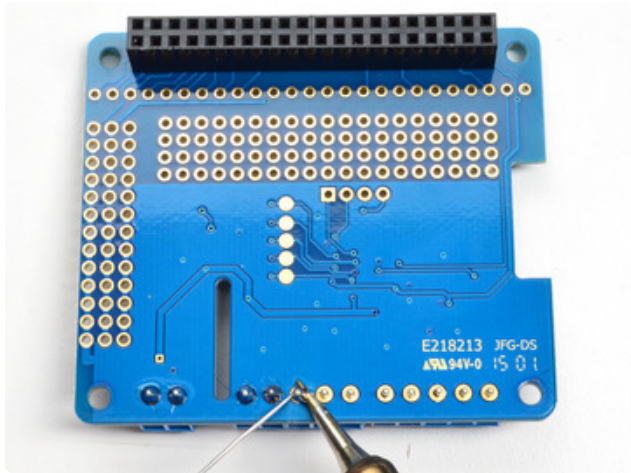
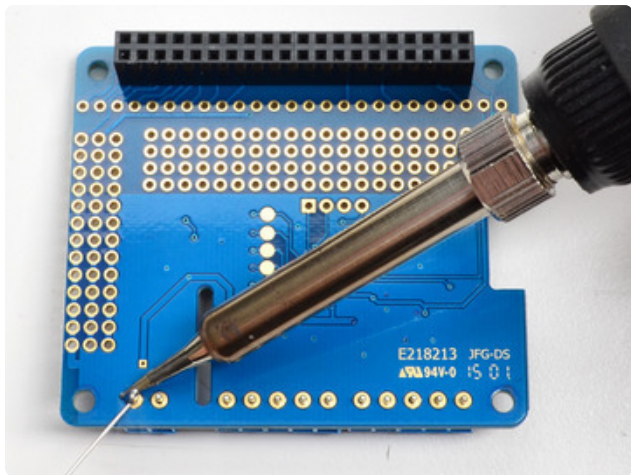
You will have 3 x 2-pin terminal blocks and 2 x 3-pin terminal blocks

Slide each of the 3-pin terminal blocks into a 2-pin to create two 5-pin blocks

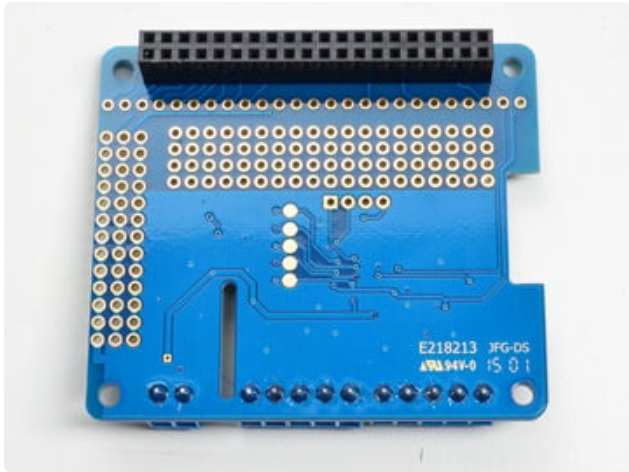


Slide the terminal blocks along the edge of the HAT, so that the 'mouth' of each block is facing out.

You can use scotch or other plain tape to keep the terminal blocks flat against the PCB while you solder



Flip over the board and solder in all of the terminal block pins



Check over your work so far, make sure each solder point is shiny, and isn't bridged or dull or cracked

You're done! You can now move onto the software side

Wiring

The wiring for the new electronics going into the bear is fairly straightforward. Make sure you have assembled the [Adafruit DC and Stepper Motor Hat \(http://adafru.it/2348\)](http://adafru.it/2348) first.

USB Cable Modification Option

To power the motors, you are going to use 5V directly from the USB Power Bank. While it could be powered from the Pi, the output isn't as high and the motors are noisy which may be heard through the speakers.

For the more compact build option, you will shorten the wiring with the following procedure.



Start with the USB to 2.1mm adapter cable.



Cut the cable with about 6 inches of wire length from the USB plug and strip the ends.



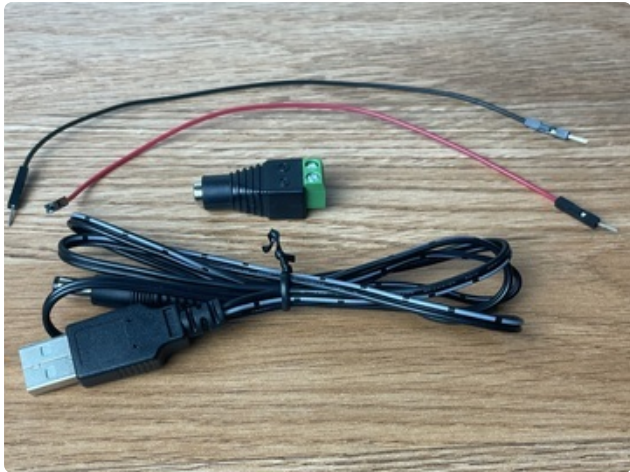
For a better connection, crimp some wire ferrules to the ends.



The side with the white stripes is the negative side.

USB Cable Adapter Option

For the easier build option, you can create a little adapter for the USB cable. This method is less compact, but non-destructive.

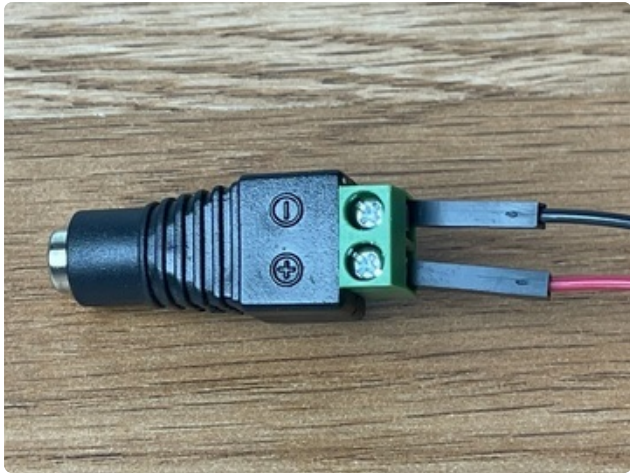


For this you will need:

1 x USB to 2.1mm adapter cable

1 x Female DC Power Adapter

2 x Jumper Wires



Install the jumper wires in the female DC power adapter. It's a good idea to use different colors so it's easier to tell which side is positive and which side is negative.



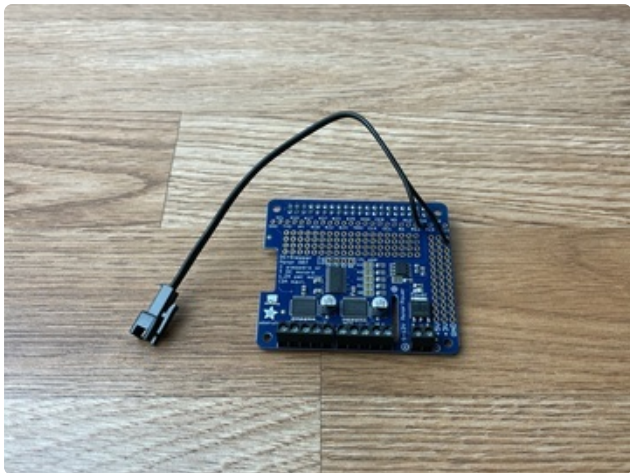
Just plug the cable into the DC power adapter and you're done.

Foot Button Connection

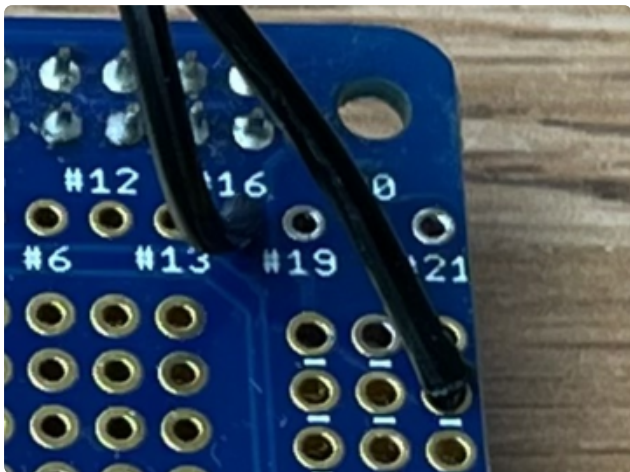
You will want to connect the other half of the JST SM connection pair to the Motor HAT. If you are soldering the white wires directly to the HAT, you should have already prepared them in the **Modifying the Bear** step.



Prepare the receptacle half of the connection by stripping and tinning the wires.

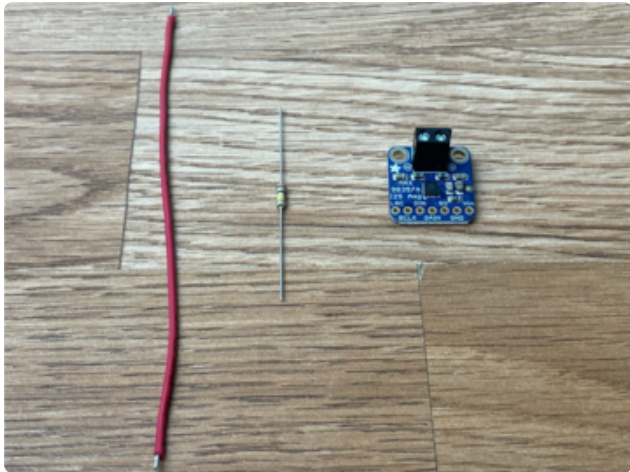


Solder the wires on the **GPIO #16** and one of the **GND** pads on the DC Motor HAT.

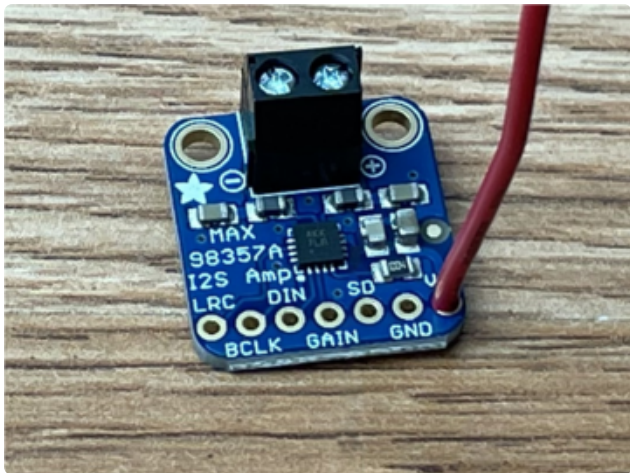


MAX98357 I2S Amplifier

Because the project is using a 3W amplifier with a 0.25W or 0.5W speaker, it's a good idea to reduce the gain as much as possible because too much can actually heat the speaker coil to the point of melting the speaker cone. This can be accomplished by connecting a 100K resistor between Gain and Vin.



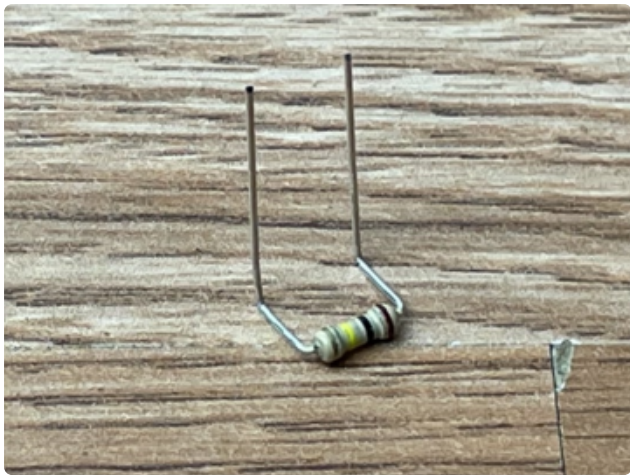
Prepare a 4-inch wire by cutting and stripping it. You can just use one of the jumper wires with both ends cut off if you don't have extra wire. You will also need a 100K resistor and the MAX98357 with only the terminal soldered on. Optionally, you could just solder the speaker directly to the MAX98357 instead of using the terminal.



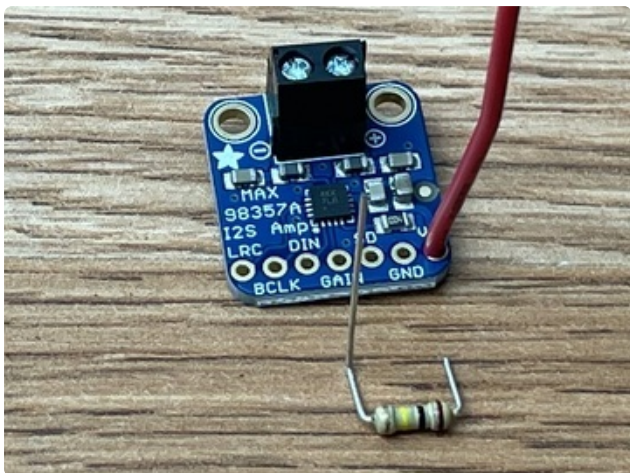
Start by soldering the wire to the **Vin** pin on the MAX98357

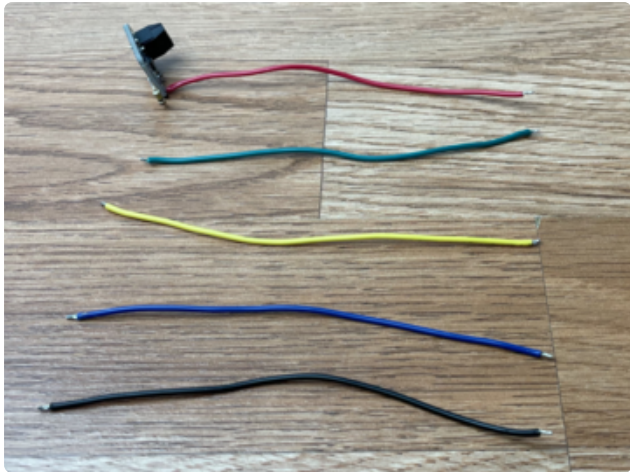


Next, take the resistor and fold the leads down at a 90 degree angle so that they are parallel to each other.

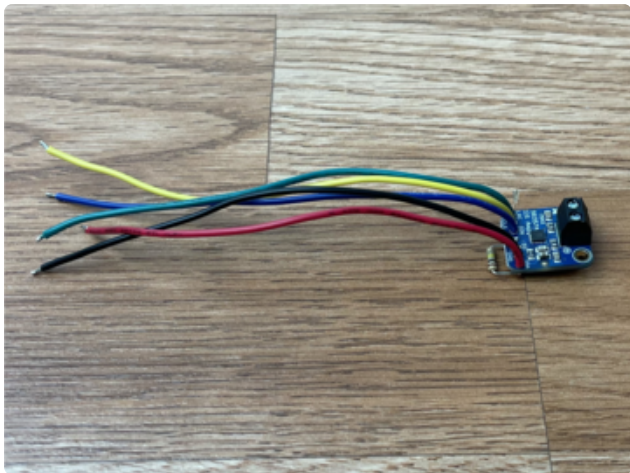


About 5mm further down from the resistor, add a 90 degree bend in the leads keeping them parallel. Cut one side of the resistor so that the cut side touches the underside of the **Vin** wire and the remaining lead can be soldered to **Gain** pin.

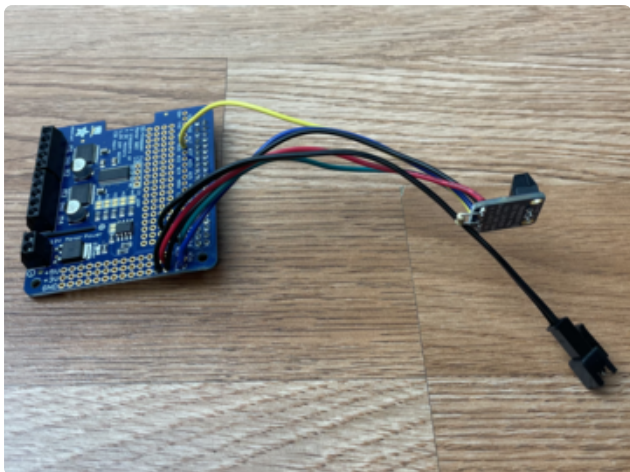




Prepare four more wires so that they are about the same length as the first wire.



Solder these wires to the **LRC**, **BCLK**, **DIN**, and **GND** pins of the MAX98357.



Solder the remaining connections according the **Circuit Diagram** page of this guide.

That's it! Everything should now be ready for assembly.

Raspberry Pi Setup

The software in this guide does not run with Bookworm at this time.

Install Raspberry Pi OS 64-bit Lite

For the bear, you will need the 64-bit version of Bullseye Raspberry Pi OS Lite because the OpenAI libraries will only install on that version. Since the project uses a voice interface, a desktop isn't required. You can refer to the [CircuitPython Libraries on Linux and Raspberry Pi \(https://adafru.it/BSN\)](https://adafru.it/BSN) guide for more help setting it up.

Once you have everything set up, you will need to open a terminal and install Blinka. Refer to the [Installing CircuitPython Libraries on Raspberry Pi \(https://adafru.it/Deo\)](https://adafru.it/Deo) page to quickly get up and running.

If you are planning on using HDMI audio, you may want to skip the Audio setup sections below since they will remove this option.

Audio System and Drivers

Before you can configure the audio devices, you will need to install some additional audio libraries:

```
sudo apt install libpulse-dev pulseaudio apulse
```

Sometimes these installations create default incompatible configuration files that cause errors. These configuration files are not necessary for the audio system to run. Run the following to remove any existing sound configuration files:

```
rm ~/.asoundrc  
sudo rm /etc/asound.conf
```

I2S Audio Setup for Compact Build Option

If you installed the MAX98357, you will need to install some software to use it. Follow the [Raspberry Pi Setup \(https://adafru.it/qpB\)](https://adafru.it/qpB) instructions from the [Adafruit MAX98357 guide \(https://adafru.it/obK\)](https://adafru.it/obK).

Afterwards follow the [Raspberry Pi Test \(https://adafru.it/18C1\)](https://adafru.it/18C1) procedure to make sure sound is playing.

USB Audio Selection for Easier Build Option

If you went with the easier build option, most likely it will just work without any changes. If not, you can tell the system which audio device to use by typing:

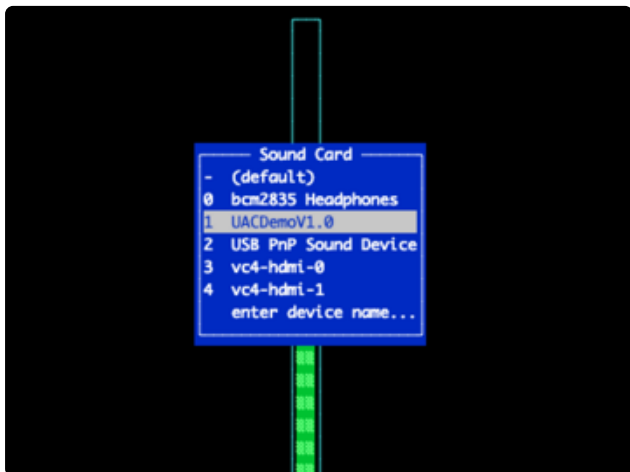
```
sudo raspi-config
```



Select **System Options**→**Audio**. Make sure **USB Audio** is selected and then tab over to **Ok** to select it. After that, you can exit to the command line and that's it.

The volume can be adjusted by running:

```
alsamixer
```



You can just adjust the **Master Volume** or if you want to adjust for the speaker only, you can press **F6** and choose **UACDemoV1.0** from the device list.

Install Required Libraries

You will need to have a few libraries installed before the script will run on your computer. Start with some prerequisite libraries:

```
sudo apt install python3-pyaudio build-essential libssl-dev libasound2 wget  
pip3 install SpeechRecognition
```


Next install the MotorKit CircuitPython library:

```
pip3 install adafruit-circuitpython-motorkit
```

Next install the OpenAI library:

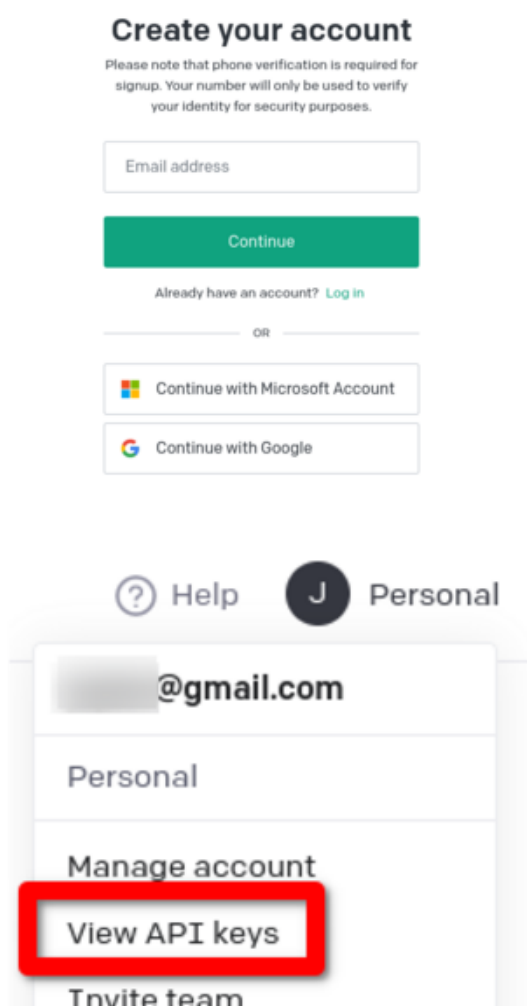
```
pip3 install --upgrade openai
```

Finally Install the Azure Speech Services library:

```
pip3 install --upgrade azure-cognitiveservices-speech
```

Create an Account with OpenAI

The OpenAI platform is managed by OpenAI and changes at their discretion, and so the details may be slightly different from what is documented here.



In your web browser, visit <https://platform.openai.com/> (<https://adafru.it/18Am>)

Click the "sign up" link. Then, you can use your e-mail to sign up, or an existing Google or Microsoft account.

OpenAI may require additional steps such as e-mail or phone verification before you can log in to your account.

Once you have completed the verification process and logged in, you will next create an API key. Use the menu in the far upper right corner (probably labeled "Personal") and then select "View API Keys".

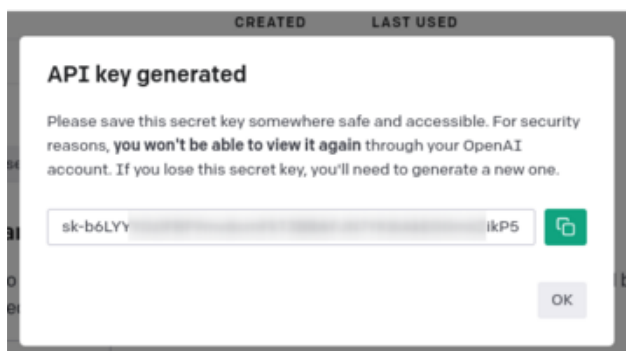
API keys

Your secret API keys are listed below. Please note that we do not display the full key after you generate them.

Do not share your API key with others, or expose it in the browser console. To protect the security of your account, OpenAI may also automatically revoke a key if it has been found has leaked publicly.

SECRET KEY	CREATED	LAST USED
sk-...9NpZ	Jul 13, 2022	Never
sk-...kWvZ	Feb 15, 2023	Mar 8, 2023

+ Create new secret key

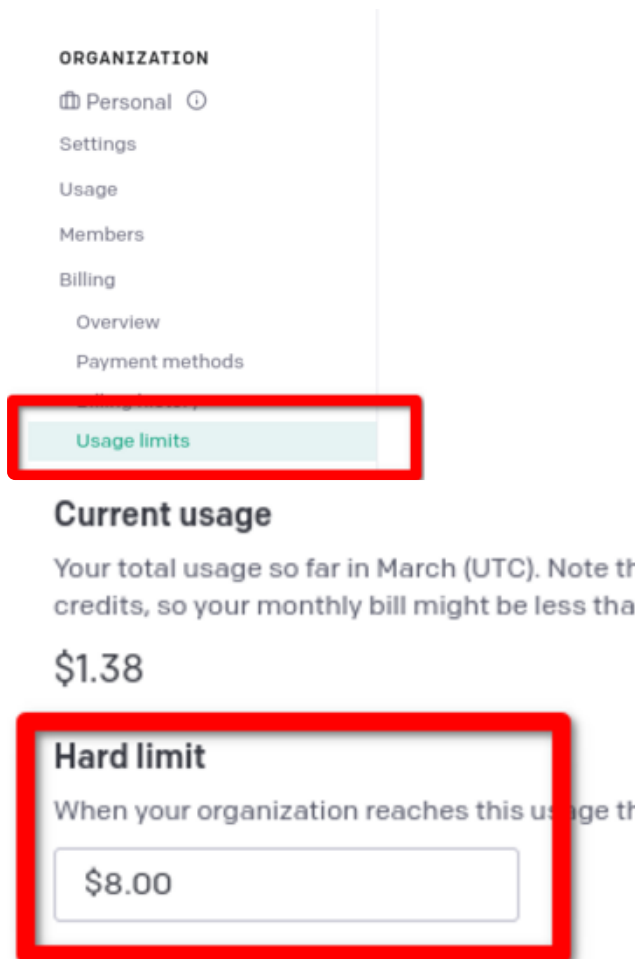


Then, create a fresh API key by clicking "Create new secret key".

Add this secret key to `/etc/environment`. From the command line, you can open it in your favorite editor and add

```
OPENAI_API_KEY="sk-b6...kP5"
```

You will need to reboot before it will be loaded into your environment variables.



At the time of writing, OpenAI provides a free credit with new accounts. After the free credit is used or expires, you'll need to enter a credit card in your billing information to keep using the service.

Using the project tends to cost a few cents per session at most, and it's easy to limit your monthly bill to a pre-set amount such as \$8.00.

To set a hard usage limit per month, visit the "Usage Limits" section of the OpenAI website.

Create an Account with Azure

The Azure platform is managed by Microsoft and changes at their discretion, and so the details may be slightly different from what is documented here.



Build in the cloud with an Azure free account

Create, deploy, and manage applications across multiple clouds, on-premises, and at the edge

Start free

Pay as you go

In your web browser, visit <https://azure.microsoft.com/> (<https://adafru.it/18C2>)

Click the **Free account** link at the top, then click the **Start free** button. Then, you can use your e-mail to sign up, or sign into an existing Microsoft account.

Microsoft may require additional steps such as e-mail verification before you can log in to your account.

Project Details

Subscription *

Resource group *

Instance Details

Region

Name *

✖ The value must not be empty.
 ✖ Only alphanumeric characters and hyphens are allowed. The value must be 2-64 characters long and cannot start or end with a hyphen.

ⓘ The free tier (FD) for this resource type is already being used by your subscription, therefore it will not appear in the dropdown below.

Pricing tier *

Once you have completed the verification process and are logged in, go to the [Speech Services](https://adafru.it/ETp) (<https://adafru.it/ETp>) section under AI + Machine Learning to create a Speech Service resource.

You will likely need to create a new Resource group. For the Instance Details, create a unique name. You can just choose a name you like and add some random numbers to the end. For Pricing Tier, there's only one choice since it's a free account.

Click **Review and Create** at the bottom.

Basics

Subscription	Azure subscription 1
Resource group	rg-1234567890
Region	East US
Name	vm-1234567890
Pricing tier	Standard S0

Network

Type	All networks, including the internet, can access this resource
------	--

Identity


Identity type	None
---------------	------


Create

< Previous

Next >

Review your settings and click **Create**.

 **Your deployment is complete**

 Deployment name: Microsoft.CognitiveServicesSpeechSe...
Subscription: [Azure subscription 1](#)
Resource group: [\[redacted\]](#)

Start time: [\[redacted\]](#)
Correlation ID: [\[redacted\]](#)

Deployment details

Next steps

[Go to resource](#)

[Give feedback](#)

[Tell us about your experience with deployment](#)

Once it is created, click the **Go to resource** Button.

Get started with your resource in Speech Studio

Try out all use cases and see other custom tools for building Speech AI models

[Go to Speech Studio](#)

Need keys and endpoint?

These keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

[Show Keys](#)

KEY 1

🔑

KEY 2

🔑

Location/Region

eastus

📍

Under overview, click the **copy button** on **Key 1** to copy it to your clipboard. You can click the **Show Keys** button as well if you want to view it.

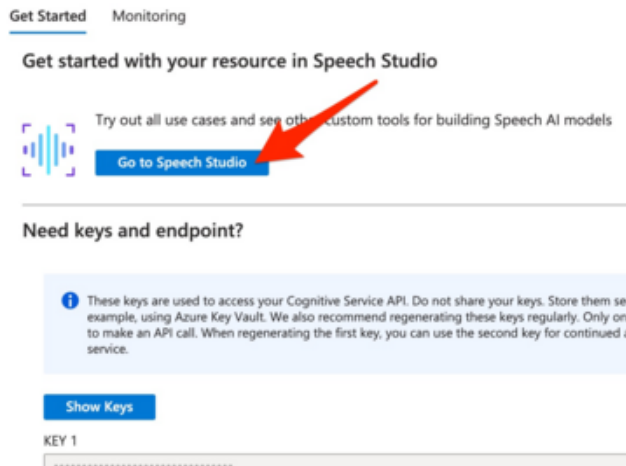
Add this secret key and the region to `/etc/environment`. From the command line, you can open it in your favorite editor and add:

```
SPEECH_KEY="4f1d...02a9"
SPEECH_REGION="eastus"
```

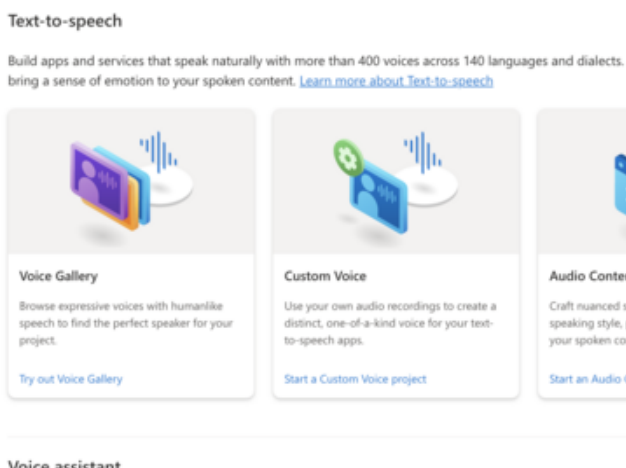
You will need to reboot before it will be loaded into your environment variables.

Using a Different Voice

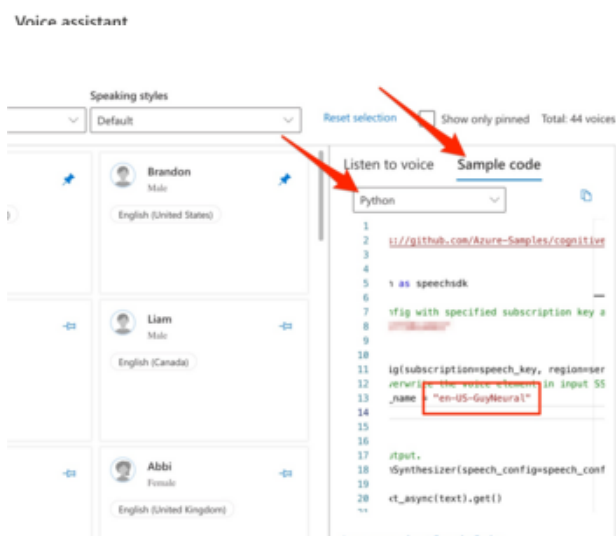
Azure Speech Services includes a number of different voices. In their Voice Gallery, you can listen to the different voices and get the Voice Name to change in your code.



Click the **Go to speech Studio** button.



Under **Text-to-speech**, click **Voice Gallery**.



You can listen to the different voices. Once you find one that you are happy with, select Sample code and change to Python.

Look for the value of `speech_config`. `speech_synthesis_voice_name`

It should be something like **en-US-GuyNeural**. You can change this in your code later.

Code the Bear

This is the code that will run on the bear. Since the API keys are stored in your environment variables, there's no need to modify the code.

```
# SPDX-FileCopyrightText: 2023 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import threading
import os
import sys

from datetime import datetime, timedelta
from queue import Queue
import time
import random
from tempfile import NamedTemporaryFile

import azure.cognitiveservices.speech as speechsdk
import speech_recognition as sr
import openai

import board
import digitalio
from adafruit_motorkit import MotorKit

# ChatGPT Parameters
SYSTEM_ROLE = (
    "You are a helpful voice assistant in the form of a talking teddy bear"
    " that answers questions and gives information"
)
CHATGPT_MODEL = "gpt-3.5-turbo"
WHISPER_MODEL = "whisper-1"

# Azure Parameters
AZURE_SPEECH_VOICE = "en-GB-OliverNeural"
DEVICE_ID = None

# Speech Recognition Parameters
ENERGY_THRESHOLD = 1000 # Energy level for mic to detect
PHRASE_TIMEOUT = 3.0 # Space between recordings for sepating phrases
RECORD_TIMEOUT = 30

# Motor Parameters
ARM_MOVEMENT_TIME = 0.5
BASE_MOUTH_DURATION = 0.2 # A higher number means slower mouth movement
SPEECH_VARIANCE = 0.1 # Higher allows more mouth movement variance.
# It pauses for BASE_MOUTH_DURATION ± SPEECH_VARIANCE
MOTOR_DUTY_CYCLE = 1.0 # Lower provides less power to the motors

# Import keys from environment variables
openai.api_key = os.environ.get("OPENAI_API_KEY")
speech_key = os.environ.get("SPEECH_KEY")
service_region = os.environ.get("SPEECH_REGION")

if openai.api_key is None or speech_key is None or service_region is None:
    print(
        "Please set the OPENAI_API_KEY, SPEECH_KEY, and SPEECH_REGION environment"
        " variables first."
    )
    sys.exit(1)

speech_config = speechsdk.SpeechConfig(subscription=speech_key,
```

```

region=service_region)
speech_config.speech_synthesis_voice_name = AZURE_SPEECH_VOICE

def sendchat(prompt):
    completion = openai.ChatCompletion.create(
        model=CHATGPT_MODEL,
        messages=[
            {"role": "system", "content": SYSTEM_ROLE},
            {"role": "user", "content": prompt},
        ],
    )
    # Send the heard text to ChatGPT and return the result
    return completion.choices[0].message.content

def transcribe(wav_data):
    # Read the transcription.
    print("Transcribing...")
    attempts = 0
    while attempts < 3:
        try:
            with NamedTemporaryFile(suffix=".wav") as temp_file:
                result = openai.Audio.translate_raw(
                    WHISPER_MODEL, wav_data, temp_file.name
                )
                return result["text"].strip()
        except (openai.error.ServiceUnavailableError, openai.error.APIError):
            time.sleep(3)
        attempts += 1
    return "I wasn't able to understand you. Please repeat that."

class Listener:
    def __init__(self):
        self.listener_handle = None
        self.recognizer = sr.Recognizer()
        self.recognizer.energy_threshold = ENERGY_THRESHOLD
        self.recognizer.dynamic_energy_threshold = False
        self.recognizer.pause_threshold = 1
        self.last_sample = bytes()
        self.phrase_time = datetime.utcnow()
        self.phrase_timeout = PHRASE_TIMEOUT
        self.phrase_complete = False
        # Thread safe Queue for passing data from the threaded recording callback.
        self.data_queue = Queue()
        self.mic_dev_index = None

    def listen(self):
        if not self.listener_handle:
            with sr.Microphone() as source:
                print(source.stream)
                self.recognizer.adjust_for_ambient_noise(source)
                audio = self.recognizer.listen(source, timeout=RECORD_TIMEOUT)
                data = audio.get_raw_data()
                self.data_queue.put(data)

    def record_callback(self, _, audio: sr.AudioData) -> None:
        # Grab the raw bytes and push it into the thread safe queue.
        data = audio.get_raw_data()
        self.data_queue.put(data)

    def speech_waiting(self):
        return not self.data_queue.empty()

    def get_speech(self):
        if self.speech_waiting():
            return self.data_queue.get()
        return None

```



```

def get_audio_data(self):
    now = datetime.utcnow()
    if self.speech_waiting():
        self.phrase_complete = False
        if self.phrase_time and now - self.phrase_time > timedelta(
            seconds=self.phrase_timeout
        ):
            self.last_sample = bytes()
            self.phrase_complete = True
            self.phrase_time = now

        # Concatenate our current audio data with the latest audio data.
        while self.speech_waiting():
            data = self.get_speech()
            self.last_sample += data

        # Use AudioData to convert the raw data to wav data.
        with sr.Microphone() as source:
            audio_data = sr.AudioData(
                self.last_sample, source.SAMPLE_RATE, source.SAMPLE_WIDTH
            )
        return audio_data

    return None

class Bear:
    def __init__(self, azure_speech_config):
        kit = MotorKit(i2c=board.I2C())
        self._arms_motor = kit.motor1
        self._mouth_motor = kit.motor2

        # Setup Foot Button
        self._foot_button = digitalio.DigitalInOut(board.D16)
        self._foot_button.direction = digitalio.Direction.INPUT
        self._foot_button.pull = digitalio.Pull.UP

        self.do_mouth_movement = False
        self._mouth_thread = threading.Thread(target=self.move_mouth, daemon=True)
        self._mouth_thread.start()
        if DEVICE_ID is None:
            audio_config =
speechsdk.audio.AudioOutputConfig(use_default_speaker=True)
        else:
            audio_config = speechsdk.audio.AudioOutputConfig(device_name=DEVICE_ID)
        self._speech_synthesizer = speechsdk.SpeechSynthesizer(
            speech_config=azure_speech_config, audio_config=audio_config
        )

        self._speech_synthesizer.synthesizing.connect(self.start_moving_mouth)
        self._speech_synthesizer.synthesis_completed.connect(self.stop_moving_mouth)

    def start_moving_mouth(self, _event):
        self.do_mouth_movement = True

    def stop_moving_mouth(self, _event):
        self.do_mouth_movement = False

    def deinit(self):
        self.do_mouth_movement = False
        self._mouth_thread.join()
        self._arms_motor.throttle = None
        self._mouth_motor.throttle = None
        self._speech_synthesizer.synthesis_started.disconnect_all()
        self._speech_synthesizer.synthesis_completed.disconnect_all()

    def move_arms_motor(self, dir_up=True):
        direction = -1 if dir_up else 1

```

```

        self._arms_motor.throttle = MOTOR_DUTY_CYCLE * direction
        time.sleep(ARM_MOVEMENT_TIME)
        # Remove Power from the motor to avoid overheating
        self._arms_motor.throttle = None

    def _move_mouth_motor(self, dir_open=True):
        duration = (
            BASE_MOUTH_DURATION
            + random.random() * SPEECH_VARIANCE
            - (SPEECH_VARIANCE / 2)
        )
        # Only power the motor while opening and let the spring close it
        self._mouth_motor.throttle = MOTOR_DUTY_CYCLE if dir_open else None
        time.sleep(duration)
        # Remove Power from the motor and let close to avoid overheating
        self._mouth_motor.throttle = None

    def foot_pressed(self):
        return not self._foot_button.value

    def move_mouth(self):
        print("Starting mouth movement thread")
        while True:
            if self.do_mouth_movement:
                self._move_mouth_motor(dir_open=True)
                self._move_mouth_motor(dir_open=False)

    def move_arms(self, hide=True):
        self._move_arms_motor(dir_up=hide)

    def speak(self, text):
        result = self._speech_synthesizer.speak_text_async(text).get()

        # Check result
        if result.reason == speechsdk.ResultReason.SynthesizingAudioCompleted:
            print("Speech synthesized for text [{}]" .format(text))
        elif result.reason == speechsdk.ResultReason.Canceled:
            cancellation_details = result.cancellation_details
            print("Speech synthesis canceled:
{}" .format(cancellation_details.reason))
            if cancellation_details.reason == speechsdk.CancellationReason.Error:
                print("Error details:
{}" .format(cancellation_details.error_details))

def main():
    listener = Listener()
    bear = Bear(speech_config)

    transcription = [""]
    bear.speak(
        "Hello there! Just give my left foot a squeeze if you would like to get my
attention."
    )
    while True:
        try:
            # If button is pressed, start listening
            if bear.foot_pressed():
                bear.speak("How may I help you?")
                listener.listen()

            # Pull raw recorded audio from the queue.
            if listener.speech_waiting():
                audio_data = listener.get_audio_data()
                bear.speak("Let me think about that")
                bear.move_arms(hide=True)
                text = transcribe(audio_data.get_wav_data())

                if text:

```

```

        if listener.phrase_complete:
            transcription.append(text)
            print(f"Phrase Complete. Sent '{text}' to ChatGPT.")
            chat_response = sendchat(text)
            transcription.append(f"> {chat_response}")
            print("Got response from ChatGPT. Beginning speech
synthesis.")

            bear.move_arms(hide=False)
            bear.speak(chat_response)
        else:
            print("Partial Phrase...")
            transcription[-1] = text

        os.system("clear")
        for line in transcription:
            print(line)
            print("", end="", flush=True)
            time.sleep(0.25)
    except KeyboardInterrupt:
        break
    bear.deinit()

if __name__ == "__main__":
    main()

```

Downloading the Code

To download the code onto your Pi, just run the following to download:

```

cd ~
wget https://raw.githubusercontent.com/adafruit/Adafruit_Learning_System_Guides/main/ChatGPT_Bear/assistant.py

```

How the Python Code Works

This is an overview about how the code works. It's mostly a combination of code to interface between:

- Azure Speech Services API
- OpenAI ChatGPT API
- OpenAI Whisper API
- SpeechRecognition Library
- CircuitPython Motor Library

The code uses threading and events in order to be able to move the mouth motors at the same time the audio output from the Azure Speech Services is playing.

Parameters

The code starts out with a number of parameters that can be altered. Here are the main parameters that you might want to change:

- The `SYSTEM_ROLE` parameter is a description to ChatGPT about how it should act.
- The `CHATGPT_MODEL` and `WHISPER_MODEL` parameters can be updated as newer versions of the API become available.
- The `AZURE_SPEECH_VOICE` is the voice name that can be altered to change how the bear sounds. See the **Create an account with Azure** page for more details on selecting a different voice in Azure Speech Services.
- The `RECORD_TIMEOUT` is the amount of time in seconds that the bear will listen after pushing the button on the foot.
- The `DEVICE_ID` is for specifying the ID of a particular Audio Output Device. See the **Troubleshooting** section on the **Usage** page of this guide to find the right value. Leaving it set to `None` will try and use the system default device.

The remaining parameters have already been tuned pretty well, but you are welcome to experiment with them.

```
# ChatGPT Parameters
SYSTEM_ROLE = (
    "You are a helpful voice assistant in the form of a talking teddy bear"
    " that answers questions and gives information"
)
CHATGPT_MODEL = "gpt-3.5-turbo"
WHISPER_MODEL = "whisper-1"

# Azure Parameters
AZURE_SPEECH_VOICE = "en-GB-OliverNeural"
DEVICE_ID = None

# Speech Recognition Parameters
ENERGY_THRESHOLD = 1000 # Energy level for mic to detect
PHRASE_TIMEOUT = 3.0 # Space between recordings for separating phrases
RECORD_TIMEOUT = 30

# Motor Parameters
ARM_MOVEMENT_TIME = 0.5
BASE_MOUTH_DURATION = 0.2 # A higher number means slower mouth movement
SPEECH_VARIANCE = 0.1 # Higher allows more mouth movement variance.
# It pauses for BASE_MOUTH_DURATION ± SPEECH_VARIANCE
MOTOR_DUTY_CYCLE = 1.0 # Lower provides less power to the motors
```

Initialization

Grab the API parameters from the environment variables and create the speech configuration.


```

openai.api_key = os.environ.get("OPENAI_API_KEY")
speech_key = os.environ.get("SPEECH_KEY")
service_region = os.environ.get("SPEECH_REGION")

speech_config = speechsdk.SpeechConfig(subscription=speech_key,
region=service_region)
speech_config.speech_synthesis_voice_name = AZURE_SPEECH_VOICE

```

OpenAI API Interface Helpers

The `sendchat` and `transcribe` functions are for packaging and sending data to the **ChatGPT** and **Whisper** APIs respectively.

```

def sendchat(prompt):
    completion = openai.ChatCompletion.create(
        model=CHATGPT_MODEL,
        messages=[
            {"role": "system", "content": SYSTEM_ROLE},
            {"role": "user", "content": prompt},
        ],
    )
    # Send the heard text to ChatGPT and return the result
    return completion.choices[0].message.content

def transcribe(wav_data):
    # Read the transcription.
    print("Transcribing...")
    attempts = 0
    while attempts < 3:
        try:
            with NamedTemporaryFile(suffix=".wav") as temp_file:
                result = openai.Audio.translate_raw(
                    WHISPER_MODEL, wav_data, temp_file.name
                )
                return result["text"].strip()
        except (openai.error.ServiceUnavailableError, openai.error.APIError):
            time.sleep(3)
        attempts += 1
    return "I wasn't able to understand you. Please repeat that."

```

The Listener Class

The `Listener` class interfaces with the SpeechRecognition library to handle listening for speech and returning the prepared audio data.

```

class Listener:
    def __init__(self):
        self.listener_handle = None
        self.recognizer = sr.Recognizer()
        self.recognizer.energy_threshold = ENERGY_THRESHOLD
        self.recognizer.dynamic_energy_threshold = False
        self.recognizer.pause_threshold = 1
        self.last_sample = bytes()
        self.phrase_time = datetime.utcnow()
        self.phrase_timeout = PHRASE_TIMEOUT
        self.phrase_complete = False
        # Thread safe Queue for passing data from the threaded recording callback.

```

```

self.data_queue = Queue()
self.mic_dev_index = None

def listen(self):
    if not self.listener_handle:
        with sr.Microphone() as source:
            print(source.stream)
            self.recognizer.adjust_for_ambient_noise(source)
            audio = self.recognizer.listen(source, timeout=RECORD_TIMEOUT)
            data = audio.get_raw_data()
            self.data_queue.put(data)

def record_callback(self, _, audio: sr.AudioData) -> None:
    # Grab the raw bytes and push it into the thread safe queue.
    data = audio.get_raw_data()
    self.data_queue.put(data)

def speech_waiting(self):
    return not self.data_queue.empty()

def get_speech(self):
    if self.speech_waiting():
        return self.data_queue.get()
    return None

def get_audio_data(self):
    now = datetime.utcnow()
    if self.speech_waiting():
        self.phrase_complete = False
        if self.phrase_time and now - self.phrase_time > timedelta(
            seconds=self.phrase_timeout
        ):
            self.last_sample = bytes()
            self.phrase_complete = True
            self.phrase_time = now

        # Concatenate our current audio data with the latest audio data.
        while self.speech_waiting():
            data = self.get_speech()
            self.last_sample += data

        # Use AudioData to convert the raw data to wav data.
        with sr.Microphone() as source:
            audio_data = sr.AudioData(
                self.last_sample, source.SAMPLE_RATE, source.SAMPLE_WIDTH
            )
        return audio_data

    return None

```

The Bear Class

The **Bear** class initializes and sets up the motors and button. It handles the timing of the motors and the speech synthesis.

One interesting function inside of this class is the **move_mouth** function which is threaded. It is in a constant loop and will constantly move the mouth while the **do_mouth_movement** variable is true. This makes turning the mouth movement on and off very easy.

In the initialization, there are also a couple of events that are triggered by the Azure Speech Services API to time the mouth movement to the speech much better. The way these events are set up is by using the `connect` function and passing in the function that will be called when triggered.

```
class Bear:
    def __init__(self, azure_speech_config):
        kit = MotorKit(i2c=board.I2C())
        self._arms_motor = kit.motor1
        self._mouth_motor = kit.motor2

        # Setup Foot Button
        self._foot_button = digitalio.DigitalInOut(board.D16)
        self._foot_button.direction = digitalio.Direction.INPUT
        self._foot_button.pull = digitalio.Pull.UP

        self.do_mouth_movement = False
        self._mouth_thread = threading.Thread(target=self.move_mouth, daemon=True)
        self._mouth_thread.start()
        if DEVICE_ID is None:
            audio_config =
speechsdk.audio.AudioOutputConfig(use_default_speaker=True)
        else:
            audio_config = speechsdk.audio.AudioOutputConfig(device_name=DEVICE_ID)
        self._speech_synthesizer = speechsdk.SpeechSynthesizer(
            speech_config=azure_speech_config, audio_config=audio_config
        )

        self._speech_synthesizer.synthesizing.connect(self.start_moving_mouth)
        self._speech_synthesizer.synthesis_completed.connect(self.stop_moving_mouth)

    def start_moving_mouth(self, _event):
        self.do_mouth_movement = True

    def stop_moving_mouth(self, _event):
        self.do_mouth_movement = False

    def deinit(self):
        self.do_mouth_movement = False
        self._mouth_thread.join()
        self._arms_motor.throttle = None
        self._mouth_motor.throttle = None
        self._speech_synthesizer.synthesis_started.disconnect_all()
        self._speech_synthesizer.synthesis_completed.disconnect_all()

    def _move_arms_motor(self, dir_up=True):
        direction = -1 if dir_up else 1
        self._arms_motor.throttle = MOTOR_DUTY_CYCLE * direction
        time.sleep(ARM_MOVEMENT_TIME)
        # Remove Power from the motor to avoid overheating
        self._arms_motor.throttle = None

    def _move_mouth_motor(self, dir_open=True):
        duration = (
            BASE_MOUTH_DURATION
            + random.random() * SPEECH_VARIANCE
            - (SPEECH_VARIANCE / 2)
        )
        # Only power the motor while opening and let the spring close it
        self._mouth_motor.throttle = MOTOR_DUTY_CYCLE if dir_open else None
        time.sleep(duration)
        # Remove Power from the motor and let close to avoid overheating
        self._mouth_motor.throttle = None

    def foot_pressed(self):
```

```

        return not self._foot_button.value

def move_mouth(self):
    print("Starting mouth movement thread")
    while True:
        if self.do_mouth_movement:
            self._move_mouth_motor(dir_open=True)
            self._move_mouth_motor(dir_open=False)

def move_arms(self, hide=True):
    self._move_arms_motor(dir_up=hide)

def speak(self, text):
    result = self._speech_synthesizer.speak_text_async(text).get()

    # Check result
    if result.reason == speechsdk.ResultReason.SynthesizingAudioCompleted:
        print("Speech synthesized for text [{}]" .format(text))
    elif result.reason == speechsdk.ResultReason.Canceled:
        cancellation_details = result.cancellation_details
        print("Speech synthesis canceled:
    {}".format(cancellation_details.reason))
        if cancellation_details.reason == speechsdk.CancellationReason.Error:
            print("Error details:
    {}".format(cancellation_details.error_details))

```

The Main Function Loop

The main function starts off by setting up instances of the **Bear** and **Listener** classes. It also sets up a transcript that it maintains throughout interactions. This allows for you to see what is being heard and what the bear is returning in a text format.

The Main Loop waits to the foot to be presses. Once pressed, it listens for up to 30 seconds or until it determines a phrase has been spoken. Once it has a phrase, it is processed and the bear moves and responds appropriately.

Once you exit the main loop, a few things are deinitialized in order to free up resources and hardware so that the bear isn't drawing excess power.

```

def main():
    listener = Listener()
    bear = Bear(speech_config)

    transcription = [""]
    bear.speak(
        "Hello there! Just give my left foot a squeeze if you would like to get my
    attention."
    )
    while True:
        try:
            # If button is pressed, start listening
            if bear.foot_pressed():
                bear.speak("How may I help you?")
                listener.listen()

            # Pull raw recorded audio from the queue.
            if listener.speech_waiting():

```



```

audio_data = listener.get_audio_data()
bear.speak("Let me think about that")
bear.move_arms(hide=True)
text = transcribe(audio_data.get_wav_data())

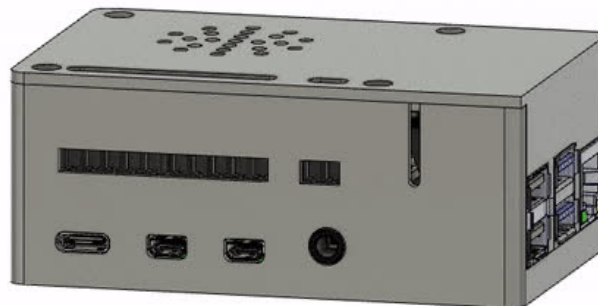
if text:
    if listener.phrase_complete:
        transcription.append(text)
        print(f"Phrase Complete. Sent '{text}' to ChatGPT.")
        chat_response = sendchat(text)
        transcription.append(f"> {chat_response}")
        print("Got response from ChatGPT. Beginning speech
synthesis.")

        bear.move_arms(hide=False)
        bear.speak(chat_response)
    else:
        print("Partial Phrase...")
        transcription[-1] = text

os.system("clear")
for line in transcription:
    print(line)
print("", end="", flush=True)
time.sleep(0.25)
except KeyboardInterrupt:
    break
bear.deinit()

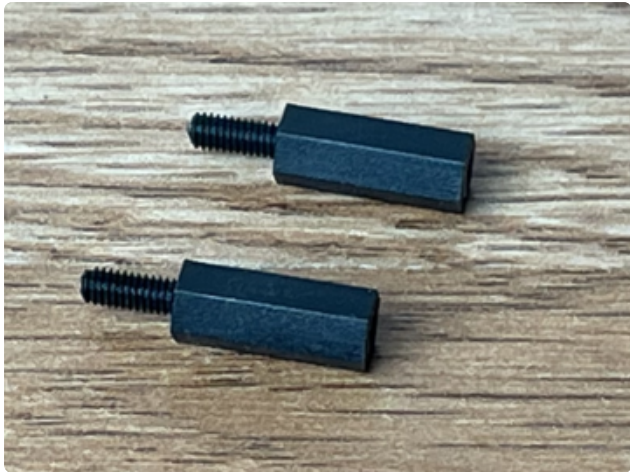
```

Assembly



Assembly Tool

The headers are much easier to install in the deep case by using a tool. If you have a 5mm Hex Nut Driver, then go ahead and use that. If not, you can make a tool out of a couple of extra nylon threaded standoffs. This tool works by having two standoffs with more friction extend into a third standoff that is being installed.



This tool just uses 2 of the 12mm threaded standoffs.



Start by completely screwing one post into another until you feel some resistance.



Continue tightening about an additional 1/6 turn until and line up the flat edges. Don't over-tighten or the plastic could break off. The idea is to have more friction between these two standoffs than you will have with the standoff you are installing.

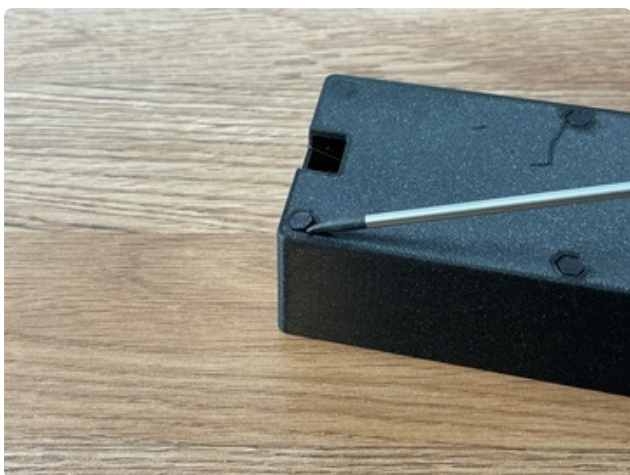


To use, just thread the standoff that you are installing onto the threads of the tool you assembled. Just use a minimal amount of friction since you want it to come off of the tool when it's installed.



Screw the standoff into a nut. When it is tight enough, just hold it in place with a finger or flathead screwdriver and unscrew the tool from the installed standoff.

Case Bottom



Make sure to remove any supports. A small flathead screwdriver or so small needle-nosed pliers should work well.



For the bottom, you'll need this hardware:

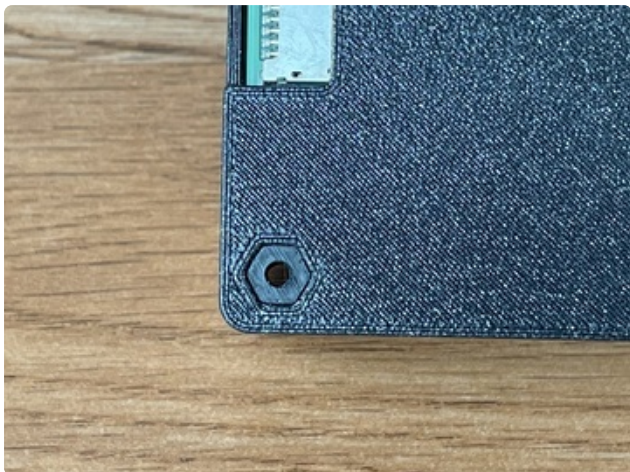
The Case Bottom

4 x 12mm nylon M2.5 standoffs

4 x nylon M2.5 nuts



Insert the Raspberry Pi 4 in the bottom case. **Make sure you don't have a MicroSD card inserted!** You'll want to have the A/V ports go in first before pushing the other side down so that it is sitting flat.



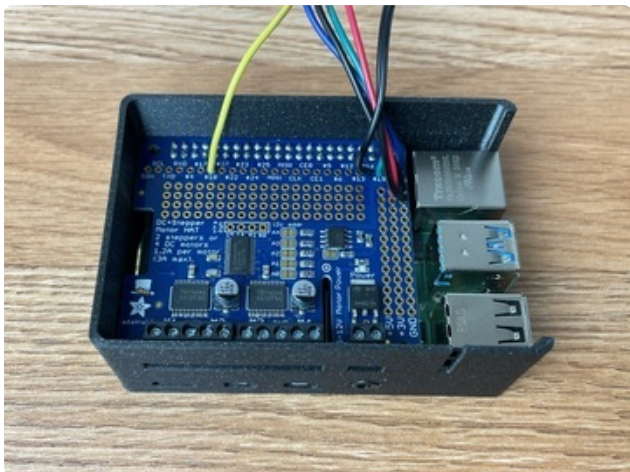
Place a nut into the hex insert on the bottom of the case.



Install a standoff through the Pi and into the nut you inserted using a tool.



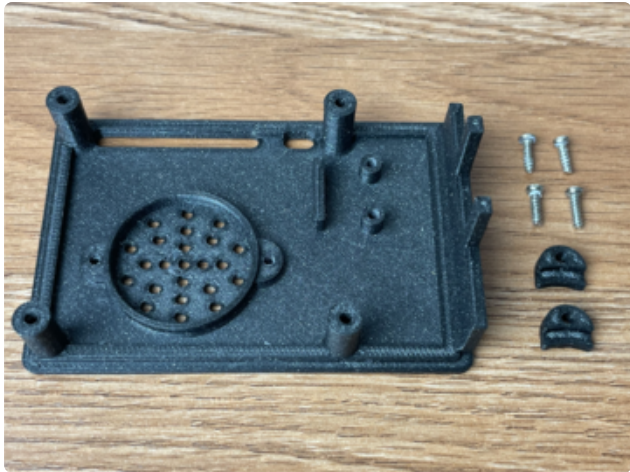
Install standoffs into the remaining corners.



Place the motor HAT onto the Pi. You can use the upper left side of the case (the corner furthest away from any ports) to easily align the GPIO headers.

Case Top

Speaker and amp installation is only required if you are doing the compact build option.



For the top, you'll need this hardware:

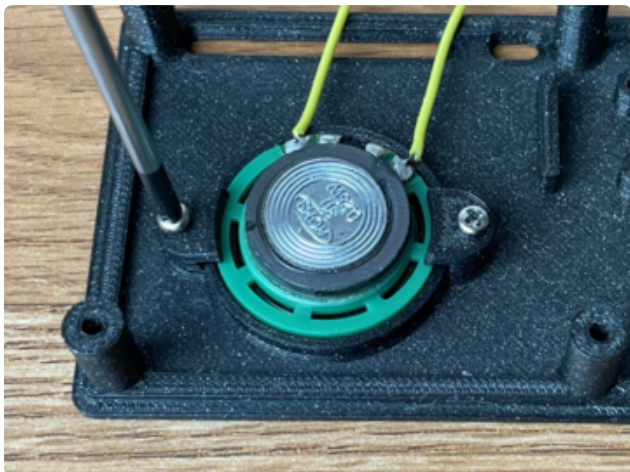
The Case Top

2 x Speaker Retainers of the appropriate size

4 x 8mm screws you set aside from disassembling the bear



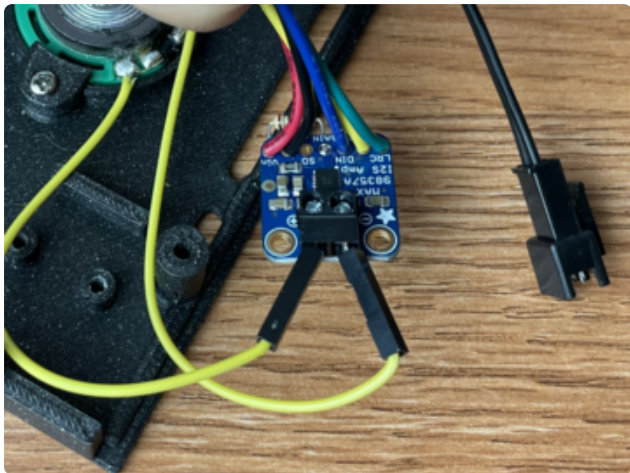
Place the speaker into the speaker holder.



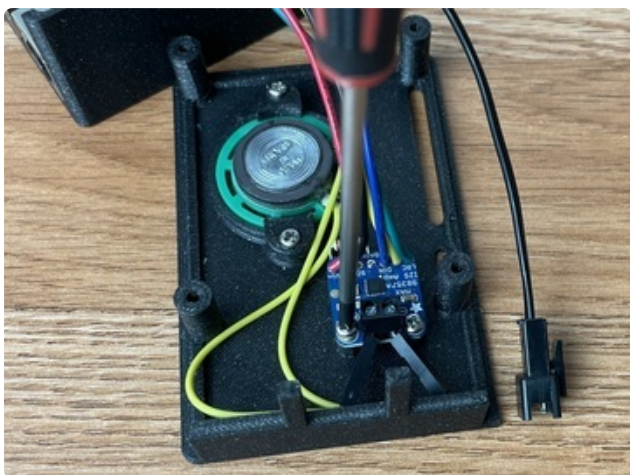
Install the speaker retainers on each side of the speaker using the screws from the bear.



The wires are a bit of a tight fit, so it's easier to install them in the MAX98357 terminal first. Place the wires in the appropriate terminal so that they have enough length once installed.

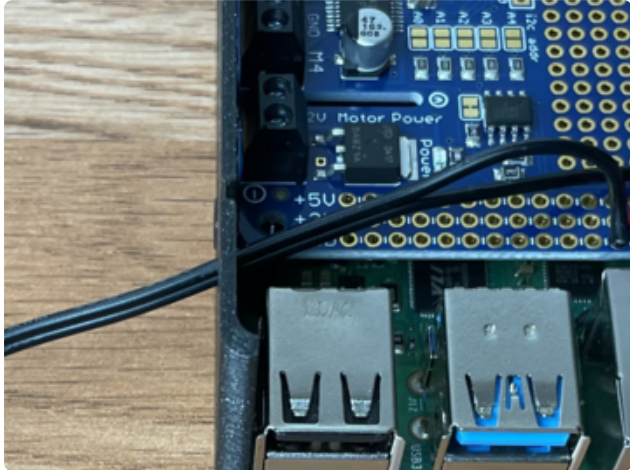


Once installed, you may need to bend the wires a little to the sides to give them enough room to fit.

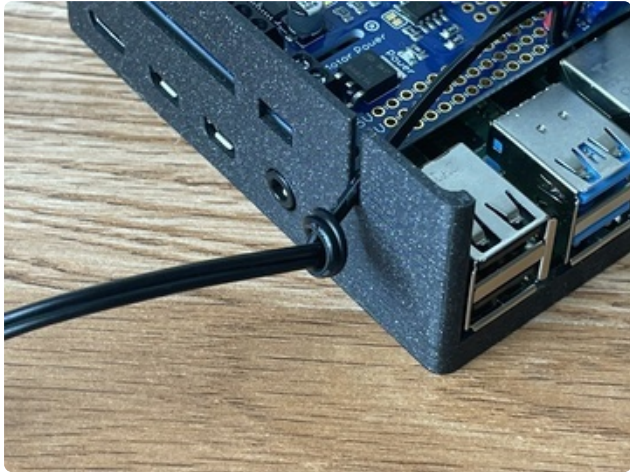


Install the MAX98357 using 2 more screws from the bear.

Final Case Assembly



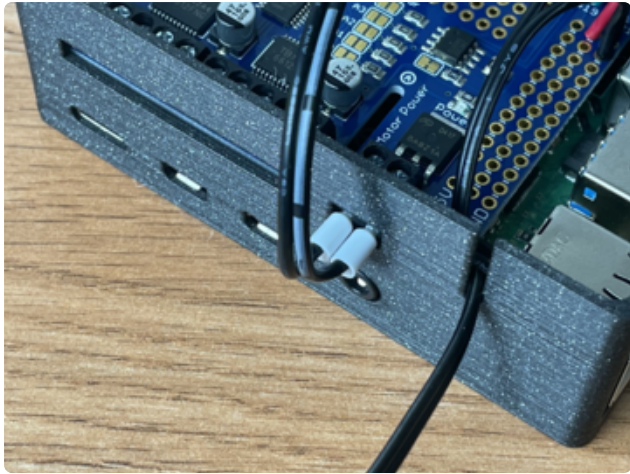
Place the wiring for the foot in the slot in the case.



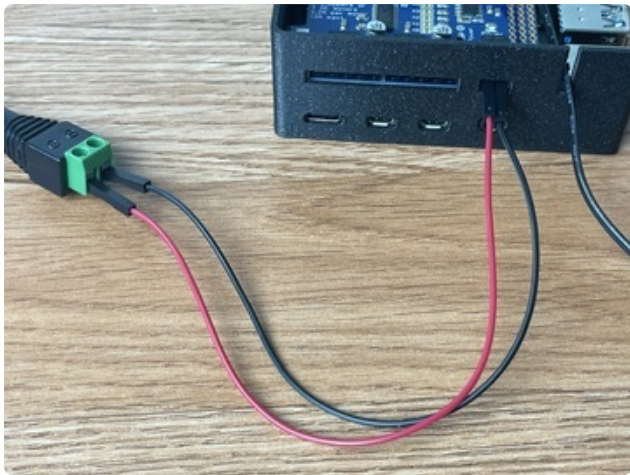
Tie a knot in the wiring close to the case, just on the outside.



Now lift up the wire and place it so the knot so it is on the inside. This will act as a strain relief.



This is a good time to install the wiring for the motor power. On the USB cable, the negative wire is the side with the white stripes. **Be sure to install correctly to avoid any damage.**



If you built an adapter, just use the jumper wires instead.



Install the lid by placing it directly over the case bottom and lowering it. If the strain relief knot is in the way, you can push it in temporarily while placing the lid. It may take a little bit of adjusting if some of the wires are in the way.



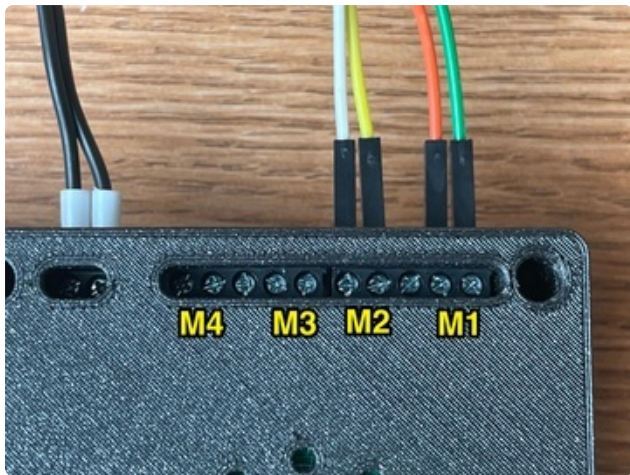
You will need four 10mm nylon screws to attach the lid.



Install the screws into the top. Only turn just enough that it feels a little snug. **It's very easy to overturn the screws and break them.**



You may also want to loosen the appropriate screws for the motor wire terminals. However, those can be installed after the case is assembled.



When you connect the wires from the bear, the wire order is important. Install the wires to the M2 and M2 terminals as shown in the photo.

USB Speaker



If you went with the easier install option, just plug the speaker into one of the USB ports of the Raspberry Pi. We'll show you how to configure it later.

USB Microphone



Be sure to install the USB microphone.



If you would like to be able to control where the sound is picked up, you could use a simple USB extension cable.

Power Connections



To power the Raspberry Pi and motors, you'll want the USB power bank along with a short USB cable. If you don't have any extras, you can use the USB cable that came with the power bank.



Connect the USB cable between one port of the power Bank and the Raspberry Pi.

Connect the other USB cable for motor power to the other port.

If the power bank turns on automatically, you can force it back off by double-pressing the power button.

Final Bear Assembly



Remove as much stuffing as you need. Depending on the build options you decided on, you may need to remove a little or a lot of the stuffing from the bear.



If you used the external speaker, place it inside the head. Keep taking out stuffing as needed. The fit is a bit tight.

If you placed your microphone on an extension, you will want to place this in the head at the same time. Try and keep it as close to the surface as possible so the stuffing doesn't muffle the sound too much.

Replace some of the stuffing back into the head afterwards until it feels like it is held in place and the head looks normal.



Place any excessively long cords into the bear next. Keep the USB plugs that go into the battery accessible.



Place the enclosure into the bear next. There's a small hole so you can see the status lights. Having that face down lets you easily know if the Pi is powered up or not. Continue adding stuffing, but leave room for the battery.



Place the battery in. Add some more stuffing. Add just enough so the body looks natural. Remember, you will need to remove the battery pack to charge it or turn it off and on.

Be careful of the plugs sticking out of the Pi. It's easy to accidentally break off connectors with everything jammed in so tight.



Finally close the flap using the velcro under the tail.



Check how well the bear sits. You may need to shift things around a bit or add or remove stuffing if it doesn't look right.

Keep in mind, the speaker in the head can make it a bit top heavy, which can cause it to fall over when the arms move. Try and keep most of the weight a bit lower if possible.

Usage



When you are ready to use the bear, open the flap on the bottom and just give it a final check to make sure the battery is charged and you have all of the cables connected. Make sure the battery and Pi are powered on.

Connect to the bear over WiFi using SSH. Once you are logged in, type the following:

```
cd ~  
python3 assistant.py
```

The bear should start talking and prompt you to squeeze its foot. Squeeze the foot and then ask it a question or request something.

Here are a few examples of requests to try:

- Tell me a joke
- Tell me a story about a robotic bear
- Tell me a poem about video game characters
- Tell me about CircuitPython

When you are finished. Just press **Control+C** a couple of times. Because of the multi-threading, sometimes it takes a moment to stop.

Troubleshooting

No Keys

Make sure you have specified your API keys before running the script. It should let you know if it wasn't able to find those values. If you set the values in `/etc/envron`, then you must reboot before the values will be loaded. If you would like to see a list of your environment variables, run the following command:

```
printenv
```

You should see a list of variables specific to your setup.

No Audio

If you hear no sound when you run the script, it could be caused by a few things. To begin with, run through the tests specific to the way you set up device.

You can also try running the following command to test the audio:

```
speaker-test
```

If you have verified that sound comes through with `speaker-test`, you may need to set the `DEVICE_ID`. To determine the Device ID, just run the following command:

```
aplay -L | grep plughw:
```

You should get a list of IDs. that begin with `plughw` like this:

```
pi@raspberrypi:~ $ aplay -L | grep plughw:
plughw:CARD=Headphones,DEV=0
plughw:CARD=vc4hdmi0,DEV=0
plughw:CARD=vc4hdmi1,DEV=0
plughw:CARD=UACDemoV10,DEV=0
pi@raspberrypi:~ $
```

For instance, if you have the USB Speaker, you should see a device like **plughw:CARD=UACDemoV10,DEV=0**. Set the **DEVICE_ID** variable to that value and it should use that.

```
DEVICE_ID = "plughw:CARD=UACDemoV10,DEV=0"
```