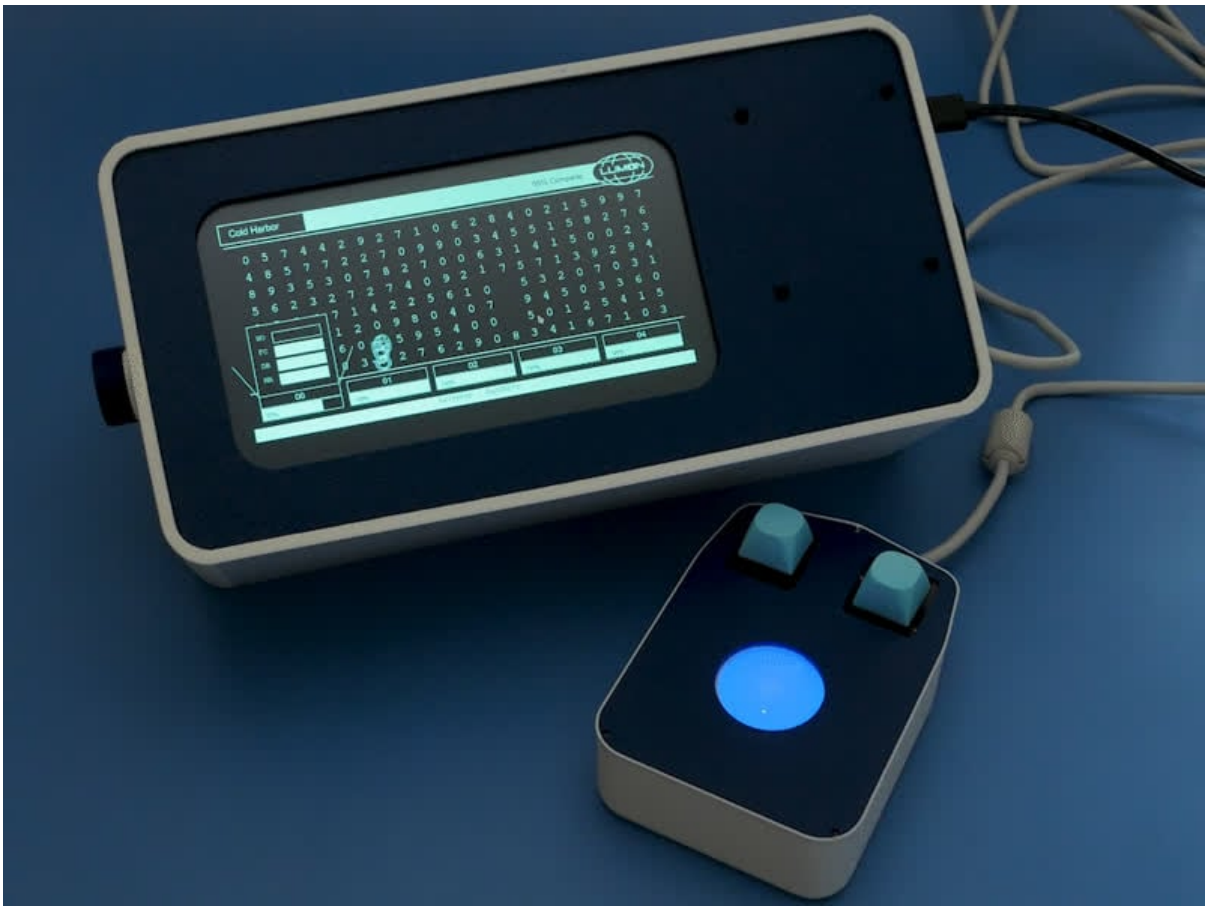
Portable Macrodata Refinement Terminal

Created by Liz Clark

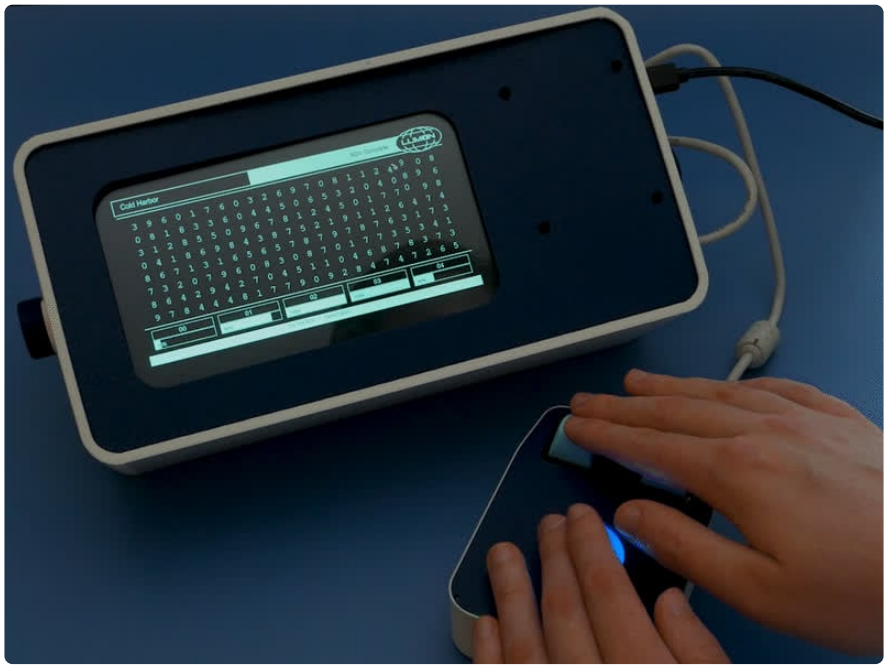


https://learn.adafruit.com/portable-macrodata-refinement-terminal

Last updated on 2025-03-05 11:56:44 AM EST

Table of Contents

Overview



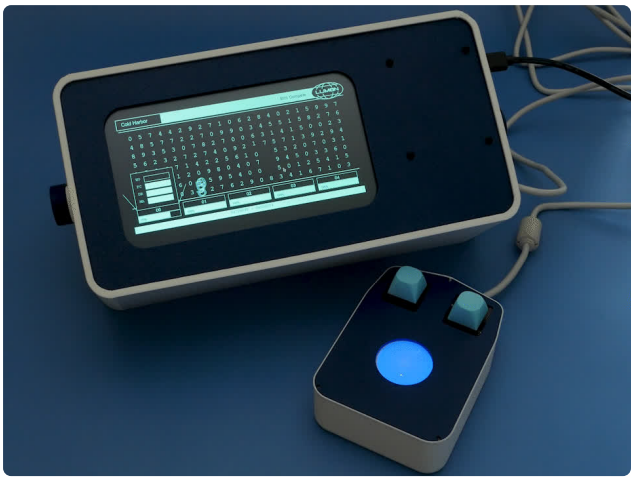
> The work is mysterious and important.

Welcome to Lumon Industries, making life better for all humankind since 1865. In your service to Lumon as a severed employee, you will partake in the Macrodata Refinement department. You will work on your newly designed terminal that allows for portability on the severed floor; a further commitment from Lumon to improve the conditions for severed employees ever since the Macrodat Uprising that your colleagues bravely organized. The next time you are granted a hall pass, feel free to carry the portable terminal by its sleek handle to your preferred area, whether it be the calming Plant Room, the inspiring Perpetuity Wing or the idyllic Mammalians Nurturable Department.

This project is built around a Raspberry Pi 5 running Python code inspired by the Macrodata Refinement program in the TV show Severance. A DRV2605L breakout vibrates a small haptic motor as your cursor gets closer to the intimidating numbers on the screen. A Raspberry Pi Touch Display 2 is used for the screen.

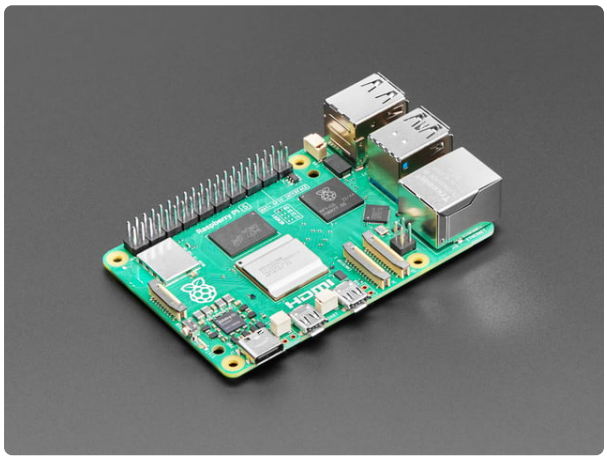
You can toggle between the bouncing screensaver and your Macrodata file with the right-click button. When you right-click, you also save your progress on your file.

A piece of acrylic gives you access to the back of your MDR terminal, letting you store your trackball and power supply for on-the-go refining during your next overtime contingency protocol or ORTBO.

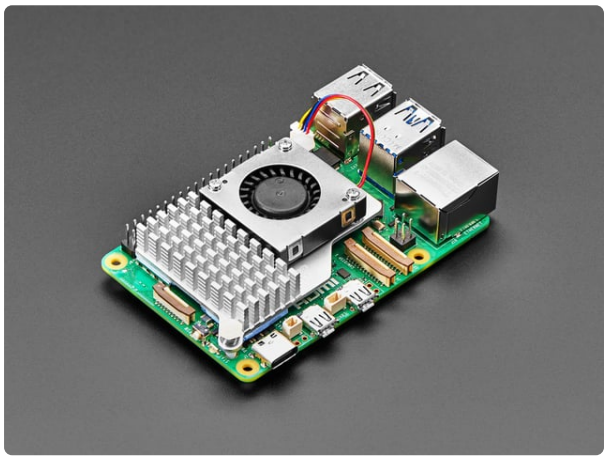


When you complete your file, you'll be invited to praise Kier with the hopes of a waffle party in your future.

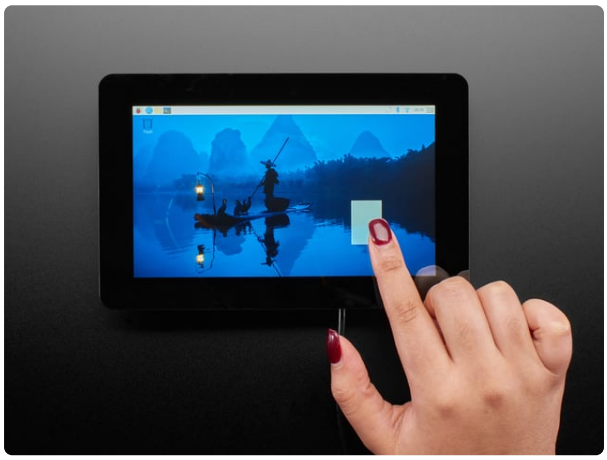
Parts



Raspberry Pi 5 - 4 GB RAM
The Raspberry Pi 5 is the newest Raspberry Pi computer, and the Pi Foundation knows you can always make a good thing better! And what could make the Pi 5 better than the...
https://www.adafruit.com/product/5812



Official Raspberry Pi 5 Active Cooler
The Raspberry Pi 5 Active Cooler is compatible with the Raspberry Pi 5 and the
https://www.adafruit.com/product/5815

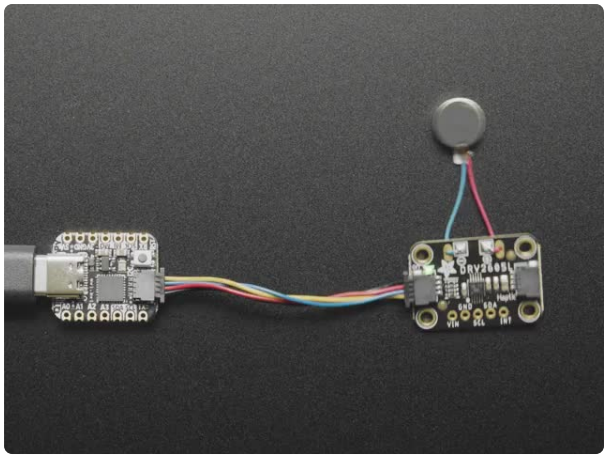


Raspberry Pi Touch Display 2 - 7" 720x1280 with Capacitive Touch
Raspberry Pi Touch Display 2 is the sequel to the popular Raspberry Pi Official Display. Like the original,...
https://www.adafruit.com/product/6079

Panel Mount Glowing Trackball

"'There is one who may claim it by right. For this assuredly is the palantír of Orthanc from the treasury of Elendil, set here by the Kings of...

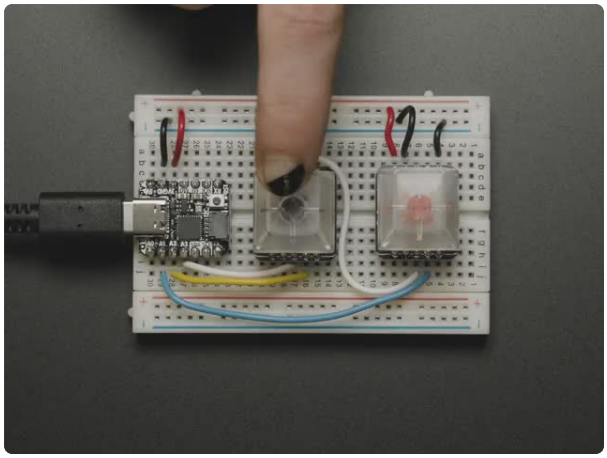
https://www.adafruit.com/product/5060



Adafruit DRV2605L Haptic Motor Controller - STEMMA QT / Qwiic

The DRV2605 from TI is a fancy little motor driver. Rather than controlling a stepper motor or DC motor, its designed specifically for controlling haptic motors -...

https://www.adafruit.com/product/2305



NeoKey Socket Breakout for Mechanical Key Switches with NeoPixel

The only thing better than a nice mechanical key, is one that also can glow any color of the rainbow - and that's what the Adafruit NeoKey Breakout will let you...

https://www.adafruit.com/product/4978

2 x Kailh Mechanical Key Switch — https://www.adafruit.com/product/5123

Clicky Blue

---

1 x Cyan MA Keycaps — https://www.adafruit.com/product/5174

for MX Compatible Switches - 5 pack

---

1 x Vibrating Mini Motor Disc — https://www.adafruit.com/product/1201

Vibrating Mini Motor Disc

Socket

1 x STEMMA QT / Qwiic JST SH 4-pin Cable — https://www.adafruit.com/product/4397
Socket

---

1 x SD/MicroSD Memory Card — https://www.adafruit.com/product/2693
16GB Class 10 - Adapter Included

---

1 x 27W PD Power Supply 5.1V 5A with USB C — https://www.adafruit.com/product/5814
Official Raspberry Pi

---

1 x Silicone Cover Stranded-Core Wire — https://www.adafruit.com/product/2051
30AWG in Various Colors

---

1 x Black Nylon Machine Screw and Stand-off Set — https://www.adafruit.com/product/3299
M2.5 Thread

---

1 x Adafruit CSI or DSI Cable Extender Thingy — https://www.adafruit.com/product/3671
Adafruit CSI or DSI Cable Extender Thingy

---

1 x Flex Cable for Raspberry Pi Camera or Display — https://www.adafruit.com/product/1648
300mm / 12"

---

1 x JST PH 2-Pin Cable — https://www.adafruit.com/product/261
Socket Connector 100mm

---

1 x JST PH 2mm 4-Pin to Pin Header Cable — https://www.adafruit.com/product/3955
200mm

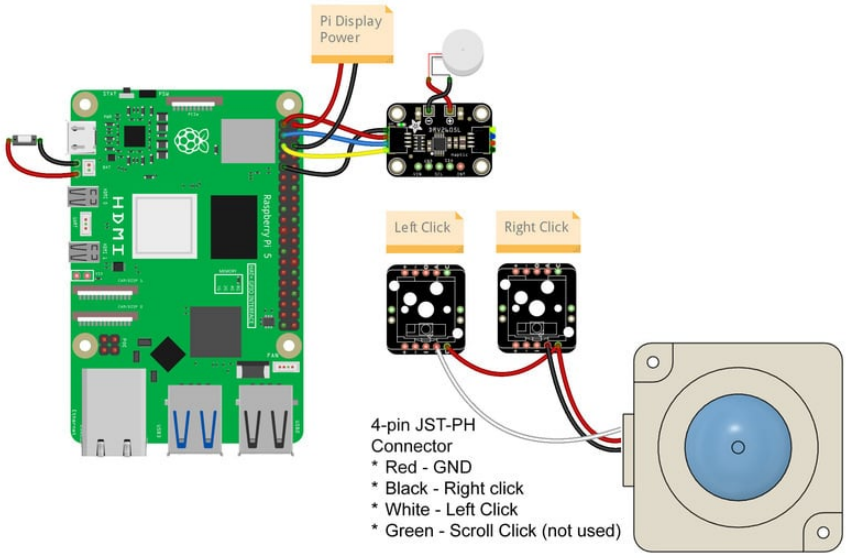
---

1 x 2.0mm Pitch Connector Kit — https://www.adafruit.com/product/4422
JST PH Compatible

---

1 x Right angle USB A extender — https://www.amazon.com/dp/B0BY8ZFGWY
Right angle USB A extender

Circuit Diagram



Raspberry Pi

- Haptic breakout GND to Pi GND (black wire)
- Haptic breakout VIN to Pi 3.3V (red wire)
- Haptic breakout SDA to Pi SDA (blue wire)
- Haptic breakout SCL to Pi SCL (yellow wire)
- Haptic breakout + to vibration motor + (red wire)
- Haptic breakout - to vibration motor - (black wire)
- Pi Display VIN to Pi 5V (red wire)
- Pi Display GND to Pi GND (black wire)
- Button input to Pi power button input (red wire)
- Button GND to Pi power button GND (black wire)

NeoKey Breakouts

- Left-click NeoKey A to JST-PH white wire
- Right-click NeoKey A to JST-PH black wire
- Left-click NeoKey C to JST-PH red wire
- Left-click NeoKey C to right-click NeoKey (red wire)

3D Printing

You can 3D print all of the parts for this project. There is a terminal chassis and a mouse enclosure inspired by the design of the Macrodata Refinement terminals on Severance. The back of the chassis can utilize a piece of acrylic or be 3D printed.

The STL files can be downloaded directly here or from Printables.

Severance_Terminal_CAD.zip

https://adafru.it/1afm

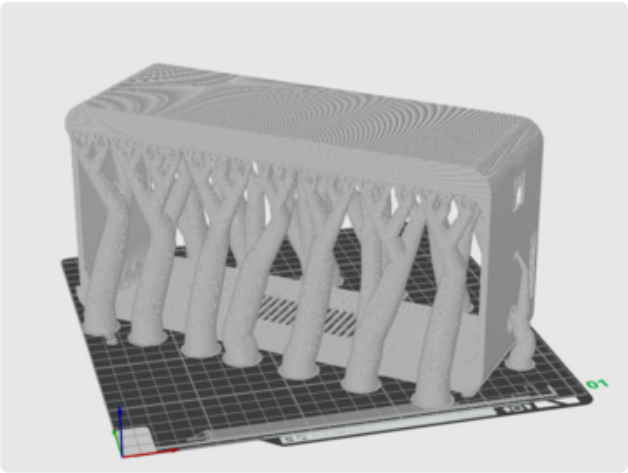
Printables Download

https://adafru.it/1afn

Parts

- Electronics mount - mount for the display, haptic breakout and Raspberry Pi. Print one, no supports.
- Terminal front plate - press fits in the front of the chassis. Print one, no supports.
- Terminal case - main enclosure for the terminal. Print one, supports required. More info below.
- Terminal back plate - slides into a track in the back of the chassis to allow for access to the electronics and storing the mouse. Can be cut from acrylic or printed with no supports.
- Terminal handle - handle for carrying and propping the chassis. Print one, build plate supports.
- Handle knobs - knobs to secure the handle to the case. Print two, no supports.

- Back plate knob - attaches to the back plate to provide a grip for sliding. Print one, no supports; color change for the knob.
- Mouse enclosure - enclosure for the trackball and NeoKey breakouts. Print one, no supports.
- Mouse mount - mount for the trackball and NeoKey breakouts. Print one, no supports.
- Mouse lid - flat lid that press fits into the mouse enclosure. Print one, no supports.



The terminal case is a really large print. You'll probably need to angle it and print it standing upright with supports to allow it to fit on your build plate if you have an average sized printer. Tree supports work really well for this print.

Acrylic

If you choose to use acrylic for the back of the case, you'll cut a piece that is 10" x 4" (25.4 cm x 10.16 cm).

Design Inspiration

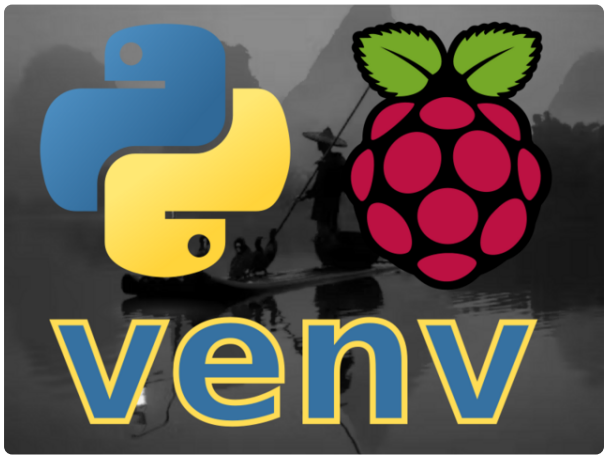
The terminal computers in Severance are inspired by the Data General 6053 terminal (https://adafru.it/1afo). A few 3D printed replicas (https://adafru.it/1afp) have been created (https://adafru.it/1afq) and I thought it would be fun to imagine a portable version that could live within the Severance universe. I took inspiration from the Sears Go Anywhere TV Radio (https://adafru.it/1afr) that were manufactured around the same time as the Data General terminal (late 1970s).

Python Virtual Environment Prep

As Carter writes in his Python Virtual Environment Usage on Raspberry Pi guide (https://adafru.it/1a9O):

> Starting with the October 10, 2023 Bookworm release (https://adafru.it/199D) of the Raspberry Pi OS (https://adafru.it/XMd), the use of Python Virtual Environments (https://adafru.it/199E) (venv) when pip installing packages is required. **No more sudo pip**. This will break things and require learning new things. Yeah.

You will need to setup a Python virtual environment (venv) on your Raspberry Pi 5 for this project. Don't worry though! If you follow along with the guide step by step, you'll be just fine.



Python Virtual Environment Usage on Raspberry Pi
By Carter Nelson
Basic Venv Usage

https://learn.adafruit.com/python-virtual-environment-usage-on-raspberry-pi/basic-venv-usage

Always venv

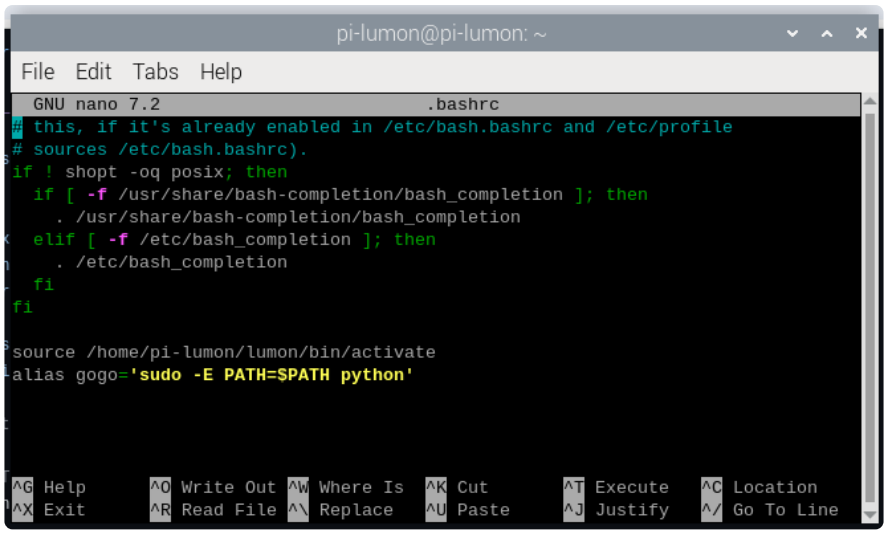
On the Other Ideas page (https://adafru.it/1a9P) in the venv guide, there is a tip for having the virtual environment enabled automatically at boot by editing the `.bashrc` file (`sudo nano .bashrc`) and adding this line to the bottom:

```
source home/user/venv/bin/activate
```

Where `venv` is the name of your virtual environment. You can take this a step further by adding the alias (https://adafru.it/1a9Q) for running Python scripts as `sudo` to your `.bashrc` file as well:

```
alias gogo='sudo -E env PATH=$PATH python'
```

You can change `gogo` to any command you want. This way, every time you boot up your Pi, you'll have your Python venv enabled and you'll be able to use your alias for running Python scripts.



Installing Blinka on Raspberry Pi

CircuitPython libraries and adafruit-blinka will work on any Raspberry Pi board! That means the original 1, the Pi 2, Pi 3, Pi 4, Pi 5, Pi Zero, Pi Zero 2 W, or even the compute modules.

At this time, Blinka requires Python version 3.7 or later, which means you will need to at least be running Raspberry Pi OS Bullseye.

Prerequisite Pi Setup!

In this page we'll assume you've already gotten your Raspberry Pi up and running and can log into the command line

Here's the quick-start for people with some experience:

1. Download the latest Raspberry Pi OS or Raspberry Pi OS Lite (https://adafru.it/Pf5) to your computer
2. Burn the OS image to your MicroSD card (https://adafru.it/dDL) using your computer
3. Re-plug the SD card into your computer (don't use your Pi yet!) and set up your wifi connection by editing supplicant.conf (https://adafru.it/yuD)
4. Activate SSH support (https://adafru.it/yuD)
5. Plug the SD card into the Pi

6. If you have an HDMI monitor we recommend connecting it so you can see that the Pi is booting OK
7. Plug in power to the Pi - you will see the green LED flicker a little. The Pi will reboot while it sets up so wait a good 10 minutes
8. If you are running Windows on your computer, install Bonjour support so you can use **.local** names, you'll need to reboot Windows after installation (https://adafru.it/lPE)
9. You can then **ssh** into **raspberrypi.local** (https://adafru.it/jvB)

The Pi Foundation has tons of guides as well (https://adafru.it/BJY)

We really really recommend the latest Raspberry Pi OS only. If you have an older Raspberry Pi OS install, run "sudo apt-get update" and "sudo apt-get upgrade" to get the latest OS!

Update Your Pi and Python

Run the standard updates:

```
sudo apt-get update
sudo apt-get -y upgrade
sudo apt-get install python3-pip
```

and upgrade setuptools:

```
sudo apt install --upgrade python3-setuptools
```

You may need to reboot prior to installing Blinka. The raspi-blinka.py script will inform you if it is necessary.

If you are installing in a virtual environment, the installer may not work correctly since it requires sudo. We recommend using pip to manually install it in that case.

Setup Virtual Environment

If you are installing on the Bookworm version of Raspberry Pi OS, you will need to install your python modules in a virtual environment. You can find more information in the Python Virtual Environment Usage on Raspberry Pi (https://adafru.it/19a5) guide.
To Install and activate the virtual environment, use the following commands:

```
sudo apt install python3-venv
python3 -m venv env --system-site-packages
```

You will need to activate the virtual environment every time the Pi is rebooted. To activate it:

```
source env/bin/activate
```

To deactivate, you can use `deactivate`, but leave it active for now.

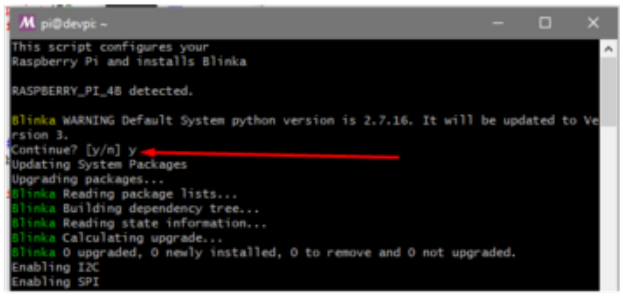
Automated Install

We put together a script to easily make sure your Pi is correctly configured and install Blinka. It requires just a few commands to run. Most of it is installing the dependencies.

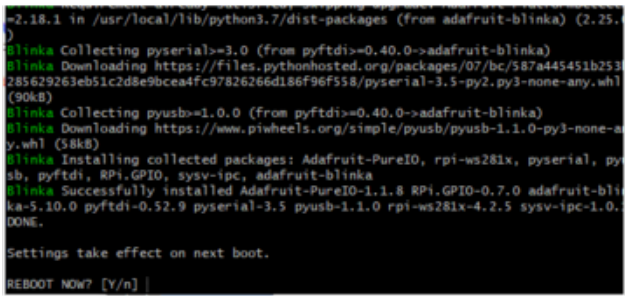
```
cd ~
pip3 install --upgrade adafruit-python-shell
wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/
master/raspi-blinka.py
sudo -E env PATH=$PATH python3 raspi-blinka.py
```

If you are installing on an earlier version such as Bullseye of Raspberry Pi OS and not using a Virtual Environment, you can call the script like:

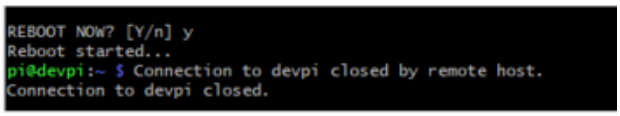
`sudo python3 raspi-blinka.py`



If you are installing an older version of Raspberry Pi OS, your system default Python is likely Python 2. If so, it will ask to confirm that you want to proceed. Choose **yes**.



It may take a few minutes to run. When it finishes, it will ask you if you would like to reboot. Choose **yes**.



Once it reboots, the connection will close. After a couple of minutes, you can reconnect.

Manual Install

If you are having trouble running the automated installation script, you can follow these steps to manually install Blinka.

Enable Interfaces

Run these commands to enable the various interfaces such as I2C and SPI:

```
sudo raspi-config nonint do_i2c 0
sudo raspi-config nonint do_spi 0
sudo raspi-config nonint do_serial_hw 0
sudo raspi-config nonint do_ssh 0
sudo raspi-config nonint do_camera 0
sudo raspi-config nonint disable_raspi_config_at_boot 0
```

Install Blinka and Dependencies

Blinka needs a few dependencies installed:

```
sudo apt-get install -y i2c-tools libgpiod-dev python3-libgpiod
pip3 install --upgrade adafruit-blinka
```

Raspberry Pi 5 Adjustments

At the moment, RPi.GPIO is installed, which causes issues. Just remove it with the following command:

```
pip3 uninstall -y RPi.GPIO
```

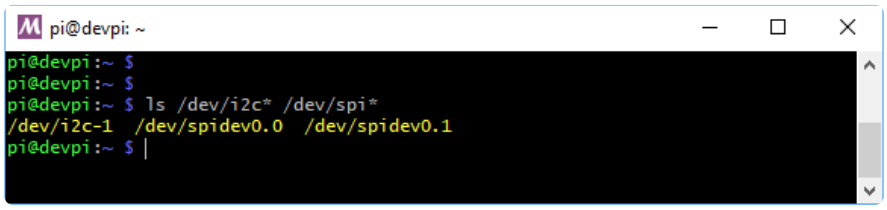
Check I2C and SPI

The script will automatically enable I2C and SPI. You can run the following command to verify:

```
ls /dev/i2c* /dev/spi*
```

You should see the response

`/dev/i2c-1 /dev/spidev0.0 /dev/spidev0.1`



Fixing CE0 and CE1 Device or Resource Busy Issue

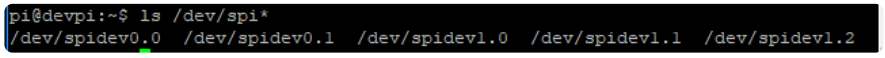
In order to use the CE0 and CE1 pins in Python, you will need to disable them from OS usage. To do so, check out the Reassigning or Disabling the SPI Chip Enable Lines (https://adafru.it/19fg) section of this guide.

Enabling Second SPI

If you are using the main SPI port for a display or something and need another hardware SPI port, you can enable it by adding the line

`dtoverlay=spi1-3cs`

to the bottom of **/boot/config.txt** and rebooting. You'll then see the addition of some /dev/spidev1.x devices:



Pi 5 : Cannot determine SOC peripheral base address

comment out this line :

```
#dtparam=spi=on
```

Blinka Test

If onewire is enabled, you may need to use another digital input besides D4.

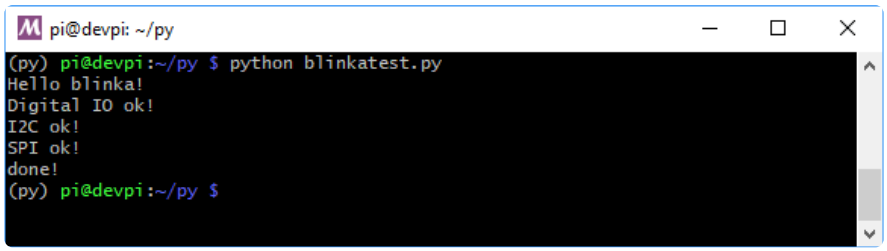
Create a new file called **blinkatest.py** with **nano** or your favorite text editor and put the following in:

```
import board
import digitalio
import busio

print("Hello, blinka!")

# Try to create a Digital input
pin = digitalio.DigitalInOut(board.D4)
print("Digital IO ok!")

# Try to create an I2C device
i2c = busio.I2C(board.SCL, board.SDA)
print("I2C ok!")

# Try to create an SPI device
spi = busio.SPI(board.SCLK, board.MOSI, board.MISO)
print("SPI ok!")

print("done!")
```

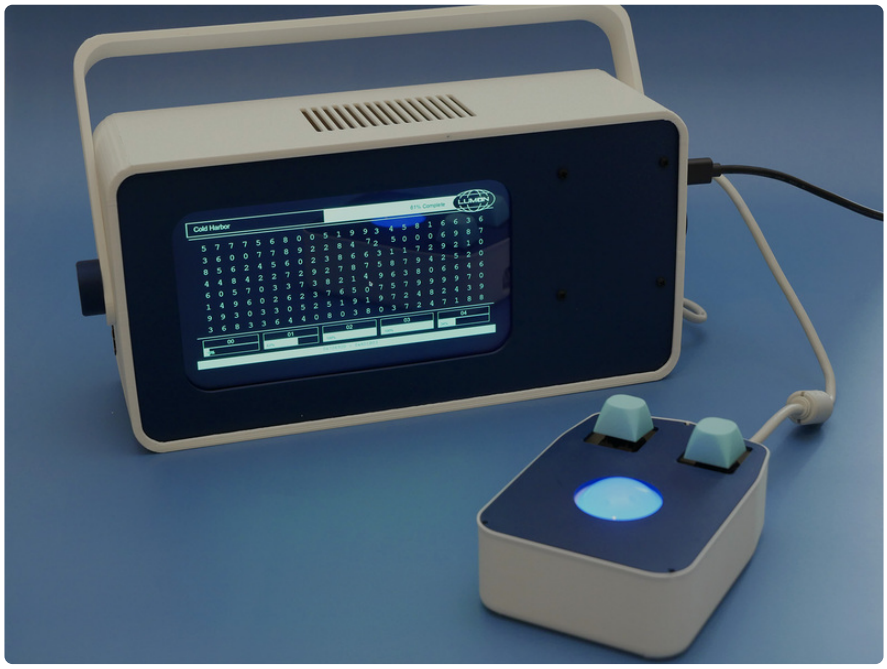
Save it and run at the command line with:

```
python3 blinkatest.py
```

You should see the following, indicating digital i/o, I2C and SPI all worked.



Terminal Software



Install the Required Libraries

You will need to install the DRV2605 library onto your Raspberry Pi. In the terminal, enter:

```
pip install adafruit-circuitpython-drv2605
```

You'll also need some extra fonts:

```
sudo apt install ttf-mscorefonts-installer
```

After installing, update the font directory cache with this command:

```
sudo fc-cache -vr
```

Download the Project Bundle

Once you've finished setting up your Raspberry Pi with Blinka and the library dependencies, you can access the Python code files and graphics by downloading the Project Bundle.

To do this, click on the **Download Project Bundle** button in the window below. It will download as a zipped folder.

```
# SPDX-FileCopyrightText: 2025 Liz Clark for Adafruit Industries
# SPDX-License-Identifier: MIT
```

```
# Based on https://github.com/Lumon-Industries/Macrodata-Refinement
# JavaScript project

import json
import os
import math
import hashlib
import shutil
import time
import random
import tkinter as tk
from PIL import Image, ImageTk, ImageFont, ImageDraw
from data import DataNumber
from data_bin import Bin
from palette import Palette
try: # blinka with haptics?
    import board
    import busio
    import adafruit_drv2605
except NotImplementedError:
    pass

TOTAL_REFINEMENT_GOAL = 250 # how many numbers to refine?
LOCATION = "Cold Harbor"

def get_username():
    username = os.environ.get('USER')
    if username:
        return username
    username = os.environ.get('LOGNAME')
    if username:
        return username
    username = os.environ.get('USERNAME')
    if username:
        return username
    return "Mark S."

def generate_serial_number(username):
    """
    Generate a Severance-style serial number based on the username.
    Format: 0xAAAAAA : 0xBBBBBB where A and B are hex values derived from username.
    """
    username_bytes = username.encode('utf-8')
    first_hex = 0
    for i in range(min(3, len(username_bytes))):
        first_hex = (first_hex << 8) | username_bytes[i]
    second_hex = 0
    if len(username_bytes) > 3:
        remaining_bytes = username_bytes[3:]
        for i, byte in enumerate(remaining_bytes):
            second_hex ^= (byte << (8 * (i % 3)))
    else:
        hash_obj = hashlib.md5(username_bytes)
        digest = hash_obj.digest()
        second_hex = int.from_bytes(digest[:3], byteorder='big')
    serial = f"0x{first_hex:06X} : 0x{second_hex:06X}"
    return serial
def calculate_distance(x1, y1, x2, y2):
    """Calculate Euclidean distance between two points"""
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# pylint: disable=too-many-branches,too-many-lines,broad-except,unused-argument
# pylint: disable=too-many-statements,too-many-locals,too-many-public-methods,too-
many-nested-blocks
class MacrodataRefinementTerminal:
    def __init__(self, username=None, location=LOCATION):
        self.palette = Palette()
        if username is None:
            username = get_username()
```

```
        # game settings
        self.username = username
        self.location = location
        self.completion = 0
        self.total_goal = TOTAL_REFINEMENT_GOAL
        self.total_refined = 0
        self.image_path = "lumon-logo.png"
        self.logo_img = Image.open("lumon-logo-small.png")

        # graphics
        self.root = tk.Tk()
        self.root.title("Lumon MDR Terminal")
        self.screen_width = self.root.winfo_screenwidth()
        self.screen_height = self.root.winfo_screenheight()
        self.root.geometry(f"{self.screen_width}x{self.screen_height}+0+0")
        self.root.grid_rowconfigure(0, weight=1)
        self.root.grid_columnconfigure(0, weight=1)
        self.root.bind("<F11>", self.toggle_fullscreen)
        self.root.bind("<Escape>", self.exit_fullscreen)
        self.root.bind("<q>", self.exit_program)
        self.root.protocol("WM_DELETE_WINDOW", self.exit_program)
        self.root.focus_force()
        self.canvas = tk.Canvas(
            self.root,
            width=self.screen_width,
            height=self.screen_height,
            bg=self.palette.BG,
            highlightthickness=0
        )
        self.canvas.pack(fill="both", expand=True)
        try:
            self.image = Image.open(self.image_path)
            self.photo = ImageTk.PhotoImage(self.image)
            self.canvas_image = self.canvas.create_image(0, 0, image=self.photo,
anchor="nw")
        except Exception as e:
            print(f"Error loading image: {e}")
            self.canvas_image = None
        self.screen = 1
        # macrodata settings
        self.completion_message_elements = {}
        self.x_pos, self.y_pos = 100, 100
        self.x_speed, self.y_speed = 2, 2
        self.base_size = 24
        self.number_spacing = 50
        self.margin = 80
        self.data_numbers = []
        self.ui_elements = {}
        self.selection_start = None
        self.selection_rect = None
        self.selection_active = False
        self.wiggle_numbers = []
        self.wiggle_timer = 0
        self.wiggle_interval = 600
        self.wiggle_amplitude = 2.0
        self.wiggle_speed = 0.2
        self.wiggle_phase = 0
        self.wiggle_phase_x = 0.0
        self.wiggle_phase_y = 0.0
        self.wiggle_phase_rotation = 0.0
        self.numbers_need_selection = False
        self.waiting_for_next_wiggle = False
        self.next_wiggle_timer = 0
        self.next_wiggle_delay = 210
        self.base_wiggle_amplitude = 1.5
        self.max_wiggle_amplitude = 10.0
        self.proximity_threshold = 350
        self.wiggle_speed_x = 0.058
        self.wiggle_speed_y = 0.047
```

```
        self.wiggle_speed_rotation = 0.02
        self.glow_step = 0
        self.fade_step = 0
        self.fade_timer = 0
        self.max_fade_steps = 20
        self.completion_photo = 0
        self.completion_triggered = False

        # mouse settings
        self.mouse_x = 0
        self.mouse_y = 0
        self.canvas.bind("<Motion>", self.track_mouse)
        self.bins = []
        self.canvas.bind("<Button-1>", self.start_selection)
        self.canvas.bind("<B1-Motion>", self.update_selection)
        self.canvas.bind("<ButtonRelease-1>", self.end_selection)
        self.root.bind("<Button-3>", self.toggle_screen)
        # start program & autosave
        self.animate()
        self.setup_autosave(interval=300)
        try:
            self.i2c = busio.I2C(board.SCL, board.SDA)
            self.drv = adafruit_drv2605.DRV2605(self.i2c)
            self.haptic_enabled = True
            self.effect_level = 0
            self.haptic_effects = {
                0: 0,
                1: 9,
                2: 13,
                3: 47,
            }
            print("Haptic feedback initialized")
        except Exception as e:
            print(f"Haptic initialization error: {e}")
            self.haptic_enabled = False

    def update_haptic_intensity(self, proximity_factor):
        if not self.haptic_enabled:
            return

        if proximity_factor < 0.1:
            self.effect_level = 0
        elif proximity_factor < 0.4:
            self.effect_level = 1
        elif proximity_factor < 0.7:
            self.effect_level = 2
        else:
            self.effect_level = 3

        if self.effect_level == 0:
            self.drv.stop()
        else:
            effect_id = self.haptic_effects[self.effect_level]
            self.drv.sequence[0] = adafruit_drv2605.Effect(effect_id)
            self.drv.play()

    def track_mouse(self, event):
        self.mouse_x = event.x
        self.mouse_y = event.y

    def apply_mouse_avoidance(self, number):
        """
        Apply an avoidance effect where numbers move away from the mouse cursor
        """
        if not hasattr(number, 'mouse_offset_x'):
            number.mouse_offset_x = 0
        if not hasattr(number, 'mouse_offset_y'):
            number.mouse_offset_y = 0
        if self.screen != 2 or number.bin_it:
```

```
            return
        dx = number.x - self.mouse_x
        dy = number.y - self.mouse_y
        distance = math.sqrt(dx*dx + dy*dy)
        avoidance_radius = 100
        max_repel_distance = 12
        if distance < avoidance_radius:
            normalized_distance = 1.0 - (distance / avoidance_radius)
            repel_factor = normalized_distance * normalized_distance * 0.8
            repel_distance = max_repel_distance * repel_factor
            if distance > 0.1:
                repel_x = (dx / distance) * repel_distance
                repel_y = (dy / distance) * repel_distance
            else:
                angle = random.uniform(0, 2 * math.pi)
                repel_x = math.cos(angle) * repel_distance
                repel_y = math.sin(angle) * repel_distance
            number.mouse_offset_x = repel_x
            number.mouse_offset_y = repel_y
        else:
            if hasattr(number, 'mouse_offset_x') and number.mouse_offset_x != 0:
                number.mouse_offset_x *= 0.95
                if abs(number.mouse_offset_x) < 0.05:
                    number.mouse_offset_x = 0
            if hasattr(number, 'mouse_offset_y') and number.mouse_offset_y != 0:
                number.mouse_offset_y *= 0.95
                if abs(number.mouse_offset_y) < 0.05:
                    number.mouse_offset_y = 0

    def save_progress(self, filepath=None):
        """
        Save the current progress and bin data to a JSON file.
        """
        if self.screen != 2 or not self.bins:
            return False
        if filepath is None:
            save_dir = f"/home/{self.username}/mdr_saves/"
            os.makedirs(save_dir, exist_ok=True)
            filepath = os.path.join(save_dir, f"mdr_{self.location.lower()}.json")
        try:
            save_data = {
                "timestamp": int(time.time()),
                "username": self.username,
                "location": self.location,
                "completion": self.completion,
                "total_goal": self.total_goal,
                "total_refined": self.total_refined,
                "bins": []
            }
            for bin_idx, bin_obj in enumerate(self.bins):
                bin_data = {
                    "bin_id": bin_idx,
                    "levels": bin_obj.levels,
                    "last_refined_time": bin_obj.last_refined_time
                }
                save_data["bins"].append(bin_data)
            with open(filepath, 'w') as f:
                json.dump(save_data, f, indent=2)
            print(f"Progress autosaved to {filepath}")
            return True
        except Exception as e:
            print(f"Error saving progress: {str(e)}")
            return False

    def load_progress(self, filepath=None):
        """
        Load progress and bin data from a JSON file.
        Verifies that the saved total goal matches the current total goal,
        otherwise starts a new save.
```

```
        """
        if filepath is None:
            filepath = os.path.join(f"/home/{self.username}/mdr_saves/",
                                    f"mdr_{self.location.lower()}.json")
        if not os.path.exists(filepath):
            print(f"Save file {filepath} not found")
            return False
        try:
            with open(filepath, 'r') as f:
                save_data = json.load(f)
            required_keys = ["timestamp", "username", "location", "completion",
"bins"]
            for key in required_keys:
                if key not in save_data:
                    print(f"Invalid save file: missing '{key}' field")
                    return False
            if "total_goal" in save_data:
                saved_goal = save_data["total_goal"]
                if saved_goal != TOTAL_REFINEMENT_GOAL:
                    print("Starting fresh with new refinement goal")
                    backup_path = filepath + f".goal_{saved_goal}.bak"
                    try:
                        shutil.copy2(filepath, backup_path)
                        print(f"Created backup of old save at {backup_path}")
                    except Exception as backup_err:
                        print(f"Warning: Could not create backup:
{str(backup_err)}")
                    return False
            self.completion = save_data["completion"]
            if "total_goal" in save_data:
                self.total_goal = save_data["total_goal"]
            else:
                self.total_goal = TOTAL_REFINEMENT_GOAL

            if "total_refined" in save_data:
                self.total_refined = save_data["total_refined"]
            if self.screen == 2 and self.bins:
                for bin_data in save_data["bins"]:
                    bin_idx = bin_data["bin_id"]
                    if bin_idx < len(self.bins):
                        self.bins[bin_idx].levels = bin_data["levels"]
                        self.bins[bin_idx].last_refined_time =
bin_data["last_refined_time"]
                        self.bins[bin_idx].update_progress_bar()
                if "total_refined" not in save_data:
                    self.update_total_refined()
                self.update_top_progress_bar()
            return True
        except Exception as e:
            print(f"Error loading progress: {str(e)}")
            return False

    def setup_autosave(self, interval=300):
        """
        Setup automatic saving at five minutes.
        """
        self.save_progress()
        self.root.after(interval * 1000, lambda: self.setup_autosave(interval))

    def create_ui_elements(self):
        """Create the UI elements for the MDR terminal"""
        for element_id in self.ui_elements.values():
            self.canvas.delete(element_id)
        self.ui_elements.clear()
        if hasattr(self, 'progress_fill_id'):
            self.canvas.delete(self.progress_fill_id)
        usable_width = self.screen_width - (2 * self.margin)
        header_height = 40
        footer_height = 30
```

```
        self.ui_elements['top_frame'] = self.canvas.create_rectangle(
            self.margin - 5, self.margin - 5,
            self.margin + usable_width -50, self.margin + header_height + 5,
            outline=self.palette.FG, fill=self.palette.BG, width=2
        )
        print(self.canvas.bbox(self.ui_elements['top_frame']))
        self.ui_elements['top_curve'] = self.canvas.create_line(
            self.margin, self.margin + header_height + 15,
            self.margin + usable_width, self.margin + header_height + 15,
            fill=self.palette.FG, width=2
        )
        self.ui_elements['location'] = self.canvas.create_text(
        self.margin + 20, self.margin + header_height/2,
        text=self.location,
        font=('Arial', 18),
        fill=self.palette.FG,
        anchor='w'
        )
        logo_x = self.margin + usable_width - 75
        logo_y = self.margin + header_height/2
        self.lumon_logo_photo = ImageTk.PhotoImage(self.logo_img) # pylint:
disable=attribute-defined-outside-init
        self.ui_elements['logo'] = self.canvas.create_image(
            logo_x, logo_y,
            image=self.lumon_logo_photo,
            anchor=tk.CENTER
        )
        logo_bbox = self.canvas.bbox(self.ui_elements['logo'])
        completion_text = f"{self.completion}% Complete"
        outlined_img = self.create_outlined_text(completion_text, font_size=20,
stroke_width=1)
        self.completion_photo = outlined_img
        completion_x = logo_bbox[0] - 20
        self.ui_elements['completion'] = self.canvas.create_image(
            completion_x, self.margin + header_height/2,
            image=outlined_img,
            anchor=tk.E
        )

        bins_y_position = self.screen_height - self.margin - 100
        progress_bar_height = 30
        spacing_after_bar = 30
        bottom_frame_y = bins_y_position + 5 + progress_bar_height +
spacing_after_bar
        self.ui_elements['bottom_curve'] = self.canvas.create_line(
            self.margin, bins_y_position - 25,
            self.margin + usable_width, bins_y_position - 25,
            fill=self.palette.FG, width=2
        )
        self.ui_elements['bottom_shield'] = self.canvas.create_rectangle(
            self.margin - 5, bottom_frame_y-16,
            self.margin + usable_width + 5, bottom_frame_y,
            outline='', fill=self.palette.BG, width=2
        )
        self.ui_elements['bottom_frame'] = self.canvas.create_rectangle(
            self.margin - 5, bottom_frame_y,
            self.margin + usable_width + 5, bottom_frame_y + footer_height,
            outline=self.palette.FG, fill=self.palette.FG, width=2
        )
        serial = generate_serial_number(self.username)
        self.ui_elements['serial'] = self.canvas.create_text(
            self.margin + usable_width/2, bottom_frame_y + footer_height/2,
            text=serial,
            font=('Courier', 14),
            fill=self.palette.BG
        )
        if 'completion' in self.ui_elements:
            self.canvas.tag_raise(self.ui_elements['completion'])
            print("Raised completion text to top")
```

```
        self.update_top_progress_bar()

    def create_outlined_text(self, text, font_size=24, stroke_width=1):
        """
        Creates an image with outlined text using PIL's stroke feature
        """
        font = ImageFont.truetype("/usr/share/fonts/truetype/msttcorefonts/
arial.ttf", font_size)
        dummy_img = Image.new("RGBA", (1, 1), (0, 0, 0, 0))
        dummy_draw = ImageDraw.Draw(dummy_img)
        bbox = dummy_draw.textbbox((0, 0), text, font=font)
        text_width = bbox[2] - bbox[0]
        text_height = bbox[3] - bbox[1]
        padding = 6
        width = text_width + padding * 2 + stroke_width * 2
        height = text_height + padding * 2 + stroke_width * 2

        img = Image.new("RGBA", (width, height), (0, 0, 0, 0))
        draw = ImageDraw.Draw(img)
        fill_color = self.palette.BG
        stroke_color = self.palette.FG
        position = (padding + stroke_width, padding)
        draw.text(position, text, font=font, fill=fill_color,
                  stroke_width=stroke_width, stroke_fill=stroke_color)
        photo = ImageTk.PhotoImage(img)
        return photo

    def create_completion_text(self, text):
        """Creates and returns a canvas image item with stroke-outlined text"""
        photo = self.create_outlined_text(text)
        self.completion_photo = photo
        return photo

    def create_bins(self):
        """Create bins at the bottom of the screen"""
        bin_count = 5
        usable_width = self.screen_width - (2 * self.margin)
        actual_bin_width = (usable_width / bin_count) * 0.9
        spacing = (usable_width - (bin_count * actual_bin_width)) / (bin_count + 1)

        for bin_obj in self.bins:
            if hasattr(bin_obj, 'visual_elements'):
                for element_id in bin_obj.visual_elements.values():
                    self.canvas.delete(element_id)
            if hasattr(bin_obj, 'level_elements'):
                for element_id in bin_obj.level_elements.values():
                    self.canvas.delete(element_id)
            if hasattr(bin_obj, 'progress_bar_elements'):
                for element_id in bin_obj.progress_bar_elements.values():
                    self.canvas.delete(element_id)
        self.bins.clear()
        bin_goal = self.total_goal // bin_count
        bins_y_position = self.screen_height - self.margin - 100
        for i in range(bin_count):
            bin_obj = Bin(actual_bin_width, i, bin_goal, self.canvas,
palette=self.palette)
            x_pos = (self.margin + spacing + (i *(actual_bin_width + spacing))
                     + (actual_bin_width / 2))
            bin_obj.x = x_pos
            bin_obj.y = bins_y_position
            self.bins.append(bin_obj)
            bin_obj.create_visual_elements()
            bin_obj.update_progress_bar()

        if 'bottom_shield' in self.ui_elements:
            self.canvas.tag_raise(self.ui_elements['bottom_shield'])
        if 'bottom_frame' in self.ui_elements:
            self.canvas.tag_raise(self.ui_elements['bottom_frame'])
        if 'serial' in self.ui_elements:
```

```
            self.canvas.tag_raise(self.ui_elements['serial'])

    def update_total_refined(self):
        """Update total refined count based on all bins' levels"""
        previous_total = self.total_refined
        self.total_refined = 0

        for bin_obj in self.bins:
            bin_total = sum(bin_obj.levels.values())
            self.total_refined += bin_total
        if self.total_refined != previous_total:
            self.update_overall_completion()

    def update_completion_text(self):
        """Updates the completion text with new percentage"""
        if 'completion' in self.ui_elements:
            completion_text = f"{self.completion}% Complete"
            outlined_img = self.create_completion_text(completion_text)
            self.canvas.itemconfig(self.ui_elements['completion'],
image=outlined_img)

    def should_update_completion(self):
        """Check if we should update the completion percentage"""
        if not self.bins:
            return False
        for bin_obj in self.bins:
            if bin_obj.show_levels or bin_obj.opening_animation or
bin_obj.closing_animation:
                return False
        return True

    def calculate_bin_percentages(self):
        """Calculate the average completion percentage across all bins"""
        if not self.bins:
            return 0
        bin_percentages = []
        for bin_obj in self.bins:
            total_levels = sum(bin_obj.levels.values())
            max_possible = bin_obj.level_goal * len(bin_obj.KEYS)
            bin_percentage = (total_levels / max_possible) * 100 if max_possible > 0
else 0
            bin_percentages.append(bin_percentage)
        avg_percentage = sum(bin_percentages) / len(bin_percentages) if
bin_percentages else 0
        return avg_percentage

    def update_overall_completion(self):
        """
        Update the overall completion percentage based on total numbers refined.
        """
        if not self.bins:
            return
        raw_completion = (self.total_refined / self.total_goal) * 100
        self.completion = int(raw_completion)
        self.update_top_progress_bar()

    def update_top_progress_bar(self):
        """
        Update the top progress bar based on bin percentages.
        """
        total_percentage = 0
        bin_count = 0
        for bin_obj in self.bins:
            total_levels = sum(bin_obj.levels.values())
            max_possible = bin_obj.level_goal * len(bin_obj.KEYS)
            bin_percentage = (total_levels / max_possible) * 100 if max_possible > 0
else 0
            total_percentage += bin_percentage
            bin_count += 1
```

```
        if bin_count > 0:
            calculated_completion = int(total_percentage / bin_count)
            self.completion = calculated_completion
        frame_bbox = self.canvas.bbox(self.ui_elements['top_frame'])
        frame_left = frame_bbox[0]
        frame_top = frame_bbox[1]
        frame_bottom = frame_bbox[3]
        logo_bbox = self.canvas.bbox(self.ui_elements['logo'])
        logo_left = logo_bbox[0]
        location_right = frame_left
        if 'location' in self.ui_elements:
            location_bbox = self.canvas.bbox(self.ui_elements['location'])
            location_right = location_bbox[2] + 20
        fill_right = logo_left + 15
        fillable_width = fill_right - location_right - 4
        fill_width = (self.completion / 100) * fillable_width
        if self.completion == 0:
            fill_left = fill_right
        else:
            fill_left = fill_right - fill_width
        fill_left = max(fill_left, location_right)
        if 'completion' in self.ui_elements:
            completion_text = f"{self.completion}% Complete"
            outlined_img = self.create_outlined_text(completion_text, font_size=20,
stroke_width=1)
            self.completion_photo = outlined_img
            self.canvas.itemconfig(self.ui_elements['completion'],
image=self.completion_photo)
            completion_x = logo_left - 20
            if 'completion' in self.ui_elements:
                self.canvas.coords(
                    self.ui_elements['completion'],
                    completion_x, self.margin + 40/2
                )
            self.canvas.tag_raise(self.ui_elements['completion'])
        if hasattr(self, 'progress_fill_id'):
            self.canvas.delete(self.progress_fill_id) # pylint: disable=access-
member-before-definition
        self.progress_fill_id = self.canvas.create_rectangle( # pylint:
disable=attribute-defined-outside-init
            fill_left, frame_top + 2,
            fill_right, frame_bottom - 2,
            fill=self.palette.FG,
            outline="",
            width=0
        )
        if 'location' in self.ui_elements:
            self.canvas.tag_raise(self.ui_elements['location'])
        if 'completion' in self.ui_elements:
            self.canvas.tag_raise(self.ui_elements['completion'])
        if 'logo' in self.ui_elements:
            self.canvas.tag_raise(self.ui_elements['logo'])

    def toggle_fullscreen(self, event=None):
        """Toggle fullscreen mode"""
        is_fullscreen = self.root.attributes("-fullscreen")
        self.root.attributes("-fullscreen", not is_fullscreen)
        return "break"

    def exit_fullscreen(self, event=None):
        """Exit fullscreen mode and quit application"""
        self.root.attributes("-fullscreen", False)
        self.exit_program()
        return "break"

    def exit_program(self, event=None):
        """Exit the program and clean up resources"""
        if hasattr(self, 'haptic_enabled') and self.haptic_enabled:
            try:
```

```
                self.drv.stop()
            except Exception as e:
                print(f"Error stopping haptic motor: {e}")
        self.root.quit()
        self.root.destroy()

    def move_logo(self):
        """Animate the logo bouncing around"""
        if self.screen == 1 and self.canvas_image:
            self.x_pos += self.x_speed
            self.y_pos += self.y_speed
            if self.x_pos + self.photo.width() >= self.screen_width or self.x_pos <=
0:
                self.x_speed = -self.x_speed
            if self.y_pos + self.photo.height() >= self.screen_height or self.y_pos
<= 0:
                self.y_speed = -self.y_speed
            self.canvas.coords(self.canvas_image, self.x_pos, self.y_pos)

    def create_number_grid(self):
        """Create the grid of numbers with proper horizontal centering"""
        for number in self.data_numbers:
            self.canvas.delete(number.text_id)
        self.data_numbers.clear()
        num_columns = 22
        num_rows = 8
        usable_width = self.screen_width - (2 * self.margin)
        header_height = 40
        bottom_line_y = self.screen_height - self.margin - 80 - 25
        total_available_height = bottom_line_y - (self.margin + header_height + 15)
        horizontal_spacing = usable_width / (num_columns + 1)
        vertical_spacing = total_available_height / (num_rows + 1)
        grid_width = horizontal_spacing * num_columns
        grid_height = vertical_spacing * num_rows
        start_x = self.margin + (usable_width - grid_width) / 2 +
horizontal_spacing/2
        start_y = self.margin + header_height + 15 + (total_available_height
                                                      - grid_height) / 2 +
vertical_spacing/2
        for row in range(num_rows):
            for col in range(num_columns):
                x = start_x + (col * horizontal_spacing)
                y = start_y + (row * vertical_spacing)
                data_number = DataNumber(x, y, self.canvas, self.base_size,
palette=self.palette)
                self.data_numbers.append(data_number)

    def update_numbers(self):
        """Update the number animations"""
        if self.screen == 2:
            for number in self.data_numbers:
                if number.bin_it:
                    number.go_bin()
                else:
                    number.go_home()

    def update_bins(self):
        """Update the bin animations and ensure proper z-ordering"""
        if self.screen == 2:
            self.update_total_refined()
            self.check_for_completion()

            for bin_obj in self.bins:
                bin_obj.update()
                if 'bottom_shield' in self.ui_elements:
                    self.canvas.tag_raise(self.ui_elements['bottom_shield'])
                if 'bottom_frame' in self.ui_elements:
                    self.canvas.tag_raise(self.ui_elements['bottom_frame'])
                if 'serial' in self.ui_elements:
```

```
                    self.canvas.tag_raise(self.ui_elements['serial'])
                for number in self.data_numbers:
                    if number.bin_it and number.bin == bin_obj:
                        if hasattr(bin_obj, 'level_elements'):
                            for element_id in bin_obj.level_elements.values():
                                self.canvas.tag_raise(number.text_id, element_id)

    def toggle_screen(self, event):
        """Toggle between logo and number screens with autosave"""
        if self.screen == 1:
            self.root.attributes("-fullscreen", True)
            self.screen = 2
            if self.canvas_image:
                self.canvas.itemconfig(self.canvas_image, state='hidden')
            self.canvas.configure(bg=self.palette.BG)
            self.completion = 0
            self.total_refined = 0
            if hasattr(self, 'completion_triggered'):
                self.completion_triggered = False
            if hasattr(self, 'completion_message_elements'):
                for element_id in self.completion_message_elements.values():
                    self.canvas.delete(element_id)
                self.completion_message_elements = {}
            self.create_ui_elements()
            self.create_bins()
            self.create_number_grid()
            self.wiggle_timer = 0
            self.waiting_for_next_wiggle = False
            self.next_wiggle_timer = 0
            load_successful = self.load_progress()
            all_bins_full = all(bin_obj.is_full() for bin_obj in self.bins)
            if all_bins_full:
                print("All bins were previously full. Resetting progress to start
over.")
                for bin_obj in self.bins:
                    bin_obj.levels = {
                        'WO': 0,
                        'FC': 0,
                        'DR': 0,
                        'MA': 0
                    }
                    bin_obj.update_progress_bar()
                self.total_refined = 0
                self.completion = 0
                self.update_top_progress_bar()
            elif not load_successful:
                self.total_refined = 0
                self.completion = 0
                for bin_obj in self.bins:
                    bin_obj.levels = {
                        'WO': 0,
                        'FC': 0,
                        'DR': 0,
                        'MA': 0
                    }
                    bin_obj.update_progress_bar()
                self.update_top_progress_bar()
            self.select_random_wiggle_group()
        else:
            self.save_progress()
            self.root.attributes("-fullscreen", True)
            self.wiggle_numbers.clear()
            self.wiggle_timer = 0
            self.waiting_for_next_wiggle = False
            self.next_wiggle_timer = 0
            if hasattr(self, 'progress_fill_id'):
                self.canvas.delete(self.progress_fill_id)
            for element_id in self.ui_elements.values():
                self.canvas.delete(element_id)
```

```
            self.ui_elements.clear()
            for bin_obj in self.bins:
                if hasattr(bin_obj, 'visual_elements'):
                    for element_id in bin_obj.visual_elements.values():
                        self.canvas.delete(element_id)
                if hasattr(bin_obj, 'level_elements'):
                    for element_id in bin_obj.level_elements.values():
                        self.canvas.delete(element_id)
                if hasattr(bin_obj, 'progress_bar_elements'):
                    for element_id in bin_obj.progress_bar_elements.values():
                        self.canvas.delete(element_id)
            self.bins.clear()
            for number in self.data_numbers:
                self.canvas.delete(number.text_id)
            self.data_numbers.clear()
            if hasattr(self, 'top_progress_elements'):
                for element_id in self.top_progress_elements.values():
                    self.canvas.delete(element_id)
                self.top_progress_elements.clear()
            if hasattr(self, 'completion_message_elements'):
                for element_id in self.completion_message_elements.values():
                    self.canvas.delete(element_id)
                self.completion_message_elements.clear()
            if self.selection_rect:
                self.canvas.delete(self.selection_rect)
                self.selection_rect = None
                self.selection_active = False
            self.screen = 1
            self.canvas.configure(bg=self.palette.BG)
            if self.canvas_image:
                self.canvas.itemconfig(self.canvas_image, state='normal')
                self.canvas.tag_raise(self.canvas_image)
                if hasattr(self, 'x_pos') and hasattr(self, 'y_pos'):
                    self.canvas.coords(self.canvas_image, self.x_pos, self.y_pos)

    def start_selection(self, event):
        """Start a selection box when mouse is pressed"""
        if self.screen == 2:
            self.selection_start = (event.x, event.y)
            self.selection_rect = self.canvas.create_rectangle(
                event.x, event.y, event.x, event.y,
                outline=self.palette.SELECT, width=2, dash=(5, 5)
            )
            self.selection_active = True

    def update_selection(self, event):
        """Update the selection box when mouse is dragged"""
        if self.screen == 2 and self.selection_active:
            x1, y1 = self.selection_start
            x2, y2 = event.x, event.y
            self.canvas.coords(self.selection_rect, x1, y1, x2, y2)

    def end_selection(self, event):
        """Process the selection when mouse is released"""
        if self.screen == 2 and self.selection_active:
            x1, y1 = self.selection_start
            x2, y2 = event.x, event.y
            selected_numbers = []
            for number in self.data_numbers:
                if number.inside(x1, y1, x2, y2) and not number.bin_it:
                    selected_numbers.append(number)
                    number.turn(self.palette.SELECT)
            wiggle_selected = [n for n in selected_numbers if n in
self.wiggle_numbers]
            non_wiggle_selected = [n for n in selected_numbers if n not in
self.wiggle_numbers]
            wiggle_capture_percent = (len(wiggle_selected) /
len(self.wiggle_numbers)
                                      if self.wiggle_numbers else 0)
```

```
            valid_selection = (
                wiggle_capture_percent >= 0.7 and
                len(non_wiggle_selected) <= len(wiggle_selected) * 2
            )
            if valid_selection:
                self.pulse_selected(wiggle_selected, 3)
                self.refine_numbers(wiggle_selected)
                for number in wiggle_selected:
                    if number in self.wiggle_numbers:
                        self.wiggle_numbers.remove(number)
                        number.needs_refinement = False
                self.waiting_for_next_wiggle = True
                self.next_wiggle_timer = 0
            else:
                for number in selected_numbers:
                    number.turn(self.palette.FG)
            self.update_overall_completion()
            self.canvas.delete(self.selection_rect)
            self.selection_active = False

    def pulse_selected(self, numbers, count, size_factor=1.5, current=0):
        """Create a pulsing animation for selected numbers"""
        if current >= count * 2:
            return
        if current < count:
            progress = current / count
            size_mod = 1.0 + (progress * (size_factor - 1.0))
        else:
            progress = (current - count) / count
            size_mod = size_factor - (progress * (size_factor - 1.0))
        for number in numbers:
            number.set_size(self.base_size * size_mod)
        self.root.after(50, lambda: self.pulse_selected(numbers, count, size_factor,
current + 1))

    def refine_numbers(self, numbers):
        """Send selected numbers to bins based on their horizontal position"""
        wiggling_numbers = [n for n in numbers if n in self.wiggle_numbers]
        if not wiggling_numbers:
            return
        DataNumber.reset_active_bin()
        all_bins_full = all(bin_obj.is_full() for bin_obj in self.bins)
        if all_bins_full:
            print("All bins are full - can't refine more numbers")
            return
        number_positions = {}
        for number in wiggling_numbers:
            target_bin = number.get_non_full_bin_for_position(self.bins)
            if target_bin:
                if target_bin not in number_positions:
                    number_positions[target_bin] = []
                number_positions[target_bin].append(number)
        selected_bin = None
        max_count = 0
        for bin_obj, bin_numbers in number_positions.items():
            if len(bin_numbers) > max_count:
                max_count = len(bin_numbers)
                selected_bin = bin_obj
        if selected_bin:
            DataNumber.active_bin = selected_bin
            for number in wiggling_numbers:
                success = number.refine(bin_obj=selected_bin)
                if not success:
                    number.needs_refinement = False
                    if number in self.wiggle_numbers:
                        self.wiggle_numbers.remove(number)

    def select_random_wiggle_group(self):
        """Randomly select a CLUSTERED group of numbers that need refinement"""
```

```
        for number in self.wiggle_numbers:
            number.needs_refinement = False
            number.wiggle_offset_x = 0
            number.wiggle_offset_y = 0
        self.wiggle_numbers.clear()
        all_bins_full = all(bin_obj.is_full() for bin_obj in self.bins)
        if all_bins_full:
            return
        if not self.data_numbers:
            return
        available_numbers = [n for n in self.data_numbers if not n.bin_it]
        if not available_numbers:
            return
        seed_number = random.choice(available_numbers)
        for number in available_numbers:
            number.distance_to_seed = calculate_distance(
                seed_number.x, seed_number.y, number.x, number.y
            )
        available_numbers.sort(key=lambda n: n.distance_to_seed)
        cluster_size = random.randint(3, 6)
        clustered_numbers = available_numbers[:min(cluster_size,
len(available_numbers))]
        self.wiggle_numbers = clustered_numbers
        for number in self.wiggle_numbers:
            number.needs_refinement = True

    def wiggle_selected_numbers(self):
        """Apply smooth, floating wiggle effect to numbers
        that need refinement and update haptics
        """
        if self.screen == 2 and self.wiggle_numbers:
            self.wiggle_phase_x += self.wiggle_speed_x
            self.wiggle_phase_y += self.wiggle_speed_y
            self.wiggle_phase_rotation += self.wiggle_speed_rotation
            proximity_factor = 0
            if len(self.wiggle_numbers) > 0:
                center_x = sum(number.x for number in
                               self.wiggle_numbers) / len(self.wiggle_numbers)
                center_y = sum(number.y for number in
                               self.wiggle_numbers) / len(self.wiggle_numbers)
                distance_to_mouse = math.sqrt((self.mouse_x - center_x)**2 +
                                              (self.mouse_y - center_y)**2)
                normalized_distance = max(0, min(1, distance_to_mouse /
self.proximity_threshold))
                proximity_factor = 1.0 - normalized_distance
                dynamic_amplitude = self.base_wiggle_amplitude + (
                    (self.max_wiggle_amplitude - self.base_wiggle_amplitude) *
                    proximity_factor**1.5
                )
                self.update_haptic_intensity(proximity_factor)
            else:
                dynamic_amplitude = self.base_wiggle_amplitude
                self.update_haptic_intensity(0)
            for number in self.wiggle_numbers:
                if not number.bin_it:
                    index = self.wiggle_numbers.index(number)
                    phase_offset = index * 0.5
                    x_freq2 = 0.37
                    y_freq2 = 0.29
                    primary_x = math.sin(self.wiggle_phase_x
                                         + phase_offset) * dynamic_amplitude
                    primary_y = math.cos(self.wiggle_phase_y
                                         + phase_offset) * dynamic_amplitude * 0.8
                    secondary_x = math.sin(self.wiggle_phase_x * x_freq2
                                           + phase_offset * 1.3) * dynamic_amplitude
* 0.3
                    secondary_y = math.cos(self.wiggle_phase_y * y_freq2
                                           + phase_offset * 0.9) * dynamic_amplitude
* 0.25
```

```
                    rot_x = math.cos(self.wiggle_phase_rotation
                                     + index * 0.7) * dynamic_amplitude * 0.2
                    rot_y = math.sin(self.wiggle_phase_rotation
                                     + index * 0.7) * dynamic_amplitude * 0.2
                    offset_x = primary_x + secondary_x + rot_x
                    offset_y = primary_y + secondary_y + rot_y
                    number.wiggle_offset_x = offset_x
                    number.wiggle_offset_y = offset_y
                    number.show_wiggle(proximity_factor)

    def check_for_completion(self):
        """Check if all bins are full and trigger completion sequence if needed"""
        if self.screen != 2:
            return
        if hasattr(self, 'completion_triggered') and self.completion_triggered:
            return
        all_bins_full = all(bin_obj.is_full() for bin_obj in self.bins)
        if self.bins:
            total_refined = sum(sum(bin_obj.levels.values()) for bin_obj in
self.bins)
            completion_pct = (total_refined / self.total_goal) * 100
        else:
            completion_pct = 0
        if all_bins_full or completion_pct >= 100:
            self.completion_sequence()

    def completion_sequence(self):
        """Start sequence for completion"""
        self.completion_triggered = True
        self.fade_out_numbers()
        self.completion = 100
        self.update_top_progress_bar()

    def fade_out_numbers(self):
        """Fade out all numbers gradually"""
        self.fade_step = 0
        self.fade_timer = 0
        self.max_fade_steps = 20
        self.animate_number_fade()

    def animate_number_fade(self):
        """Animate the fading out of all numbers"""
        if self.fade_step >= self.max_fade_steps:
            self.show_completion_message()
            return
        alpha = int(255 * (1 - (self.fade_step / self.max_fade_steps)))
        for number in self.data_numbers:
            number.alpha = alpha
            number.update_display()
        self.fade_step += 1
        self.root.after(50, self.animate_number_fade)

    def show_completion_message(self):
        """Show the completion celebration message"""
        if hasattr(self, 'completion_message_elements'):
            for element_id in self.completion_message_elements.values():
                self.canvas.delete(element_id)
        self.completion_message_elements = {}
        if 'top_curve' in self.ui_elements:
            top_y = self.canvas.coords(self.ui_elements['top_curve'])[1] + 15
        else:
            top_y = self.margin + 80
        if 'bottom_curve' in self.ui_elements:
            bottom_y = self.canvas.coords(self.ui_elements['bottom_curve'])[1] - 15
        else:
            bottom_y = self.screen_height - self.margin - 130
        center_x = self.screen_width / 2
        center_y = (top_y + bottom_y) / 2
        self.completion_message_elements['percent'] = self.canvas.create_text(
```

```
            center_x, center_y - 30,
            text="100%",
            font=('Courier', 48, 'bold'),
            fill=self.palette.FG,
            anchor='center'
        )
        self.completion_message_elements['praise'] = self.canvas.create_text(
            center_x, center_y + 30,
            text="Praise Kier",
            font=('Courier', 36, 'bold'),
            fill=self.palette.FG,
            anchor='center'
        )
        self.glow_step = 0
        self.animate_completion_glow()

    def animate_completion_glow(self):
        """Create a subtle glowing/pulsing effect for the completion message"""
        if (not hasattr(self, 'completion_message_elements')
            or 'percent' not in self.completion_message_elements):
            return
        glow_factor = 0.8 + (0.2 * (math.sin(self.glow_step / 10) + 1) / 2)
        percent_size = int(48 * glow_factor)
        praise_size = int(36 * glow_factor)
        self.canvas.itemconfig(
            self.completion_message_elements['percent'],
            font=('Courier', percent_size, 'bold')
        )
        self.canvas.itemconfig(
            self.completion_message_elements['praise'],
            font=('Courier', praise_size, 'bold')
        )
        self.glow_step += 1
        self.root.after(100, self.animate_completion_glow)

    def animate(self):
        """Main animation loop"""
        if self.screen == 1:
            self.root.attributes("-fullscreen", True)
            self.move_logo()
        else:
            self.update_bins()
            self.update_numbers()
            for number in self.data_numbers:
                self.apply_mouse_avoidance(number)
            if self.waiting_for_next_wiggle:
                self.next_wiggle_timer += 1
                self.next_wiggle_delay = random.randint(180, 240)
                if self.next_wiggle_timer >= self.next_wiggle_delay:
                    self.waiting_for_next_wiggle = False
                    self.next_wiggle_timer = 0
                    self.select_random_wiggle_group()
            elif (not self.wiggle_numbers and not self.waiting_for_next_wiggle
                  and self.wiggle_timer == 0):
                self.select_random_wiggle_group()
            self.wiggle_selected_numbers()
        self.root.after(20, self.animate)

    def run(self):
        """Start the application"""
        self.root.mainloop()

if __name__ == "__main__":
    app = MacrodataRefinementTerminal()
    app.run()
```

After downloading the Project Bundle, move the folder to your **/home/user** directory. Then, unzip the folder by **right-clicking** on the folder in the File Manager and selecting **Extract** or with your preferred command line tool. Keep the following files in the **/home/user** directory:

- **lumon.py**
- **data.py**
- **data_bins.py**
- **palette.py**
- **lumon-logo.png**
- **lumon-logo-small.png**

How the Code Works

The code consists of four files. Two of the files are helper files: **data.py** file handles the number grid graphics and **data_bins.py** handles the bin animations and data. **palette.py** contains the color palette for the entire program. The main program file is in **lumon.py**. It handles the gameplay, graphics, mouse input and saving/loading the game progress JSON files.

Special Features

The code is involved since it is basically a full game, but here are some fun features in it:

- Your username on your system is converted to the serial number at the bottom of the screen (0x000000 : 0x000000)
- You can change the goal number for numbers to refine at the top if you want to experience 100% faster
- JSON files are used to save and load progress
- As you move the cursor, the numbers will move out of its path
- As the cursor gets closer to the chosen numbers, the chosen numbers will jump around more and the vibration from the haptic motor will increase in intensity
- Since its written in CPython, it doesn't have to be run on a Raspberry Pi. Most of the development was actually done on Windows. You'll just need to edit some of the file paths to make it fit your system.

Code References

It could be that the Macrodata Refinement Terminal is the new PipBoy for maker prop recreations. I referenced a few open source code iterations when working on this Python version. LumonMDR by andrewchilicki (https://adafru.it/1afq) is written in C and

Macrodata-Refinement by Lumon-Industries (https://adafru.it/1afs) is written in Javascript.

Run on Boot

You probably don't want to be futzing with a full keyboard and terminal window to launch this project every time. Luckily you can configure it to run on boot by creating a `systemd` unit file that runs the Python script within the Python venv after the desktop environment loads.

File Permissions

To allow for all of the files to run on boot without issue, you'll need to adjust the file permissions before running the `systemd` file. Run these commands in the terminal:

```
chmod +x /home/USERNAME/lumon.py
chmod +x /home/USERNAME/data.py
chmod +x /home/USERNAME/data_bins.py
chmod +x /home/USERNAME/palette.py
```

Replace `USERNAME` with your username on your Raspberry Pi.

Then, run this command for the folder that your saved JSON files are stored:

```
chmod 755 /home/USERNAME/mdr_saves/
```

systemd

In a terminal, use a text editor to create a systemd file called `mdr`:

```
sudo nano /lib/systemd/system/mdr.service
```

Copy and paste the contents of the text file below into the systemd file:

```
[Unit]
Description=Macrodata Refinement
After=multi-user.service

[Service]
Type=simple
PAMName=login
User=pi-lumon
Group=pi-lumon
WorkingDirectory=/home/pi-lumon
Environment=DISPLAY=:0
Environment=XAUTHORITY=/home/pi-lumon/.Xauthority
ExecStart=/home/pi-lumon/lumon/bin/python /home/pi-lumon/lumon.py
Restart=on-failure
RestartSec=5s
```

```
[Install]
WantedBy=multi-user.target
```

Replace "pi-lumon" with the username on your Raspberry Pi. The default is usually "pi".

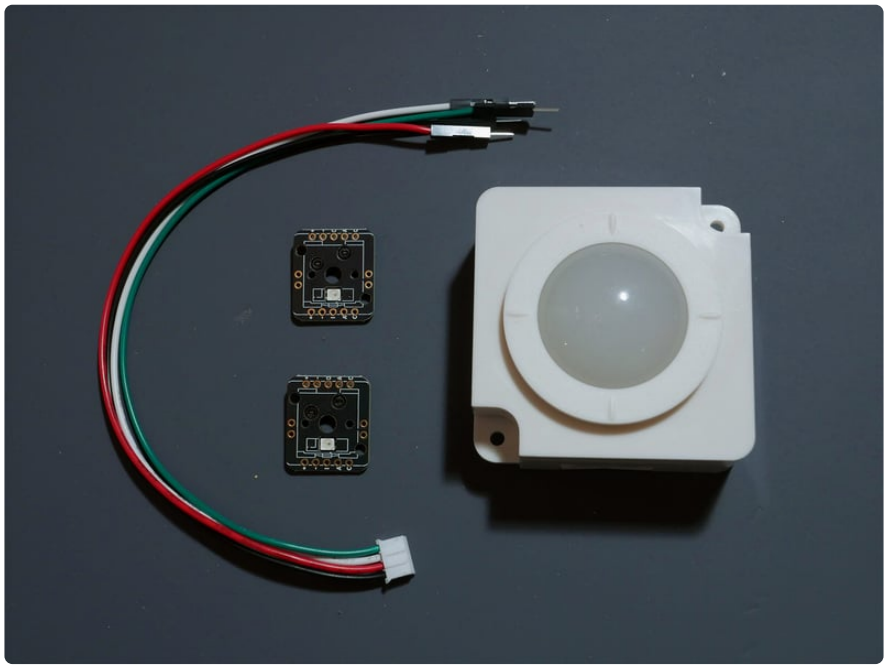
The file waits for the desktop environment to load and then launches the Python script within the virtual environment. If it fails, it will keep trying every 5 seconds.

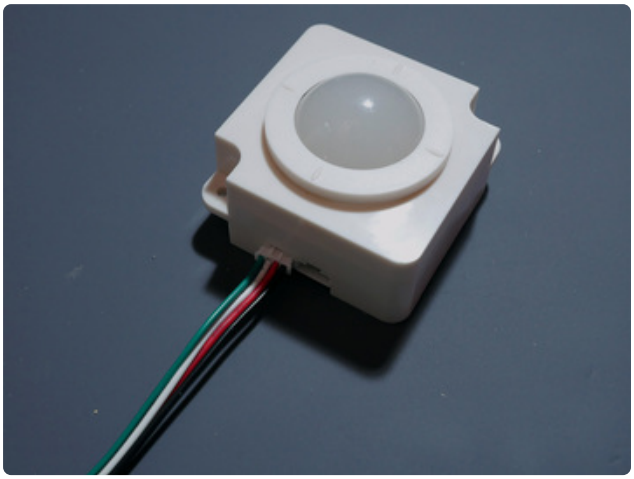
Save the file. Then, in a terminal run these commands:

```
sudo systemctl enable mdr
sudo systemctl start mdr
```

This enables the service and starts it. To test, reboot your Pi. After the desktop environment loads, you should see the script startup automatically with the bouncing Lumon logo.

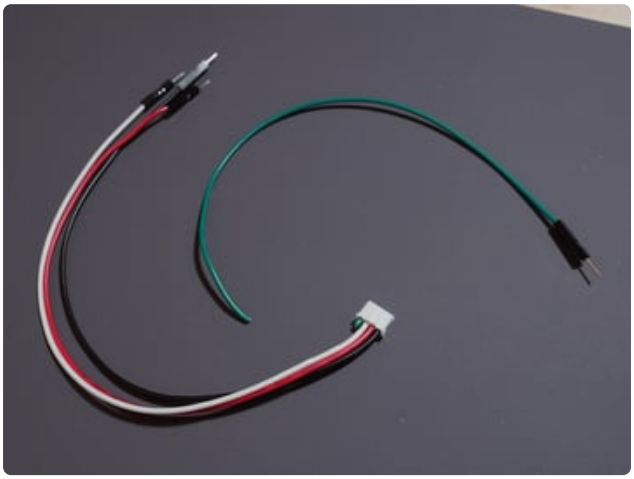
Soldering





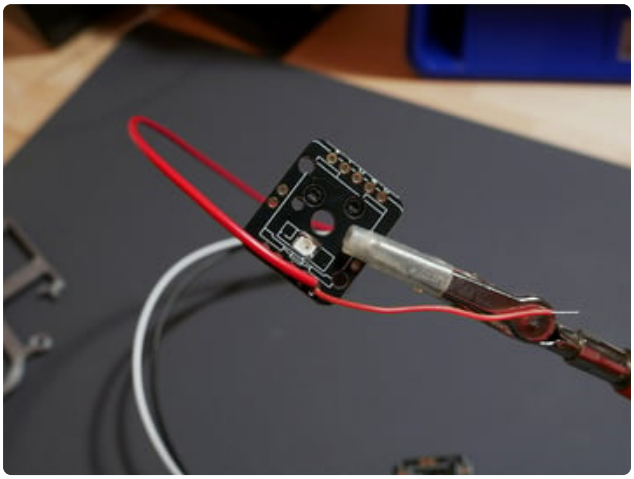
The 4-pin JST-PH cables are a match for the trackball mouse button input. This input lets you add left-click, right-click and scroll-click buttons. The cable pinout is:

Black - right-click
Red - GND
White - left-click
Green - scroll-click

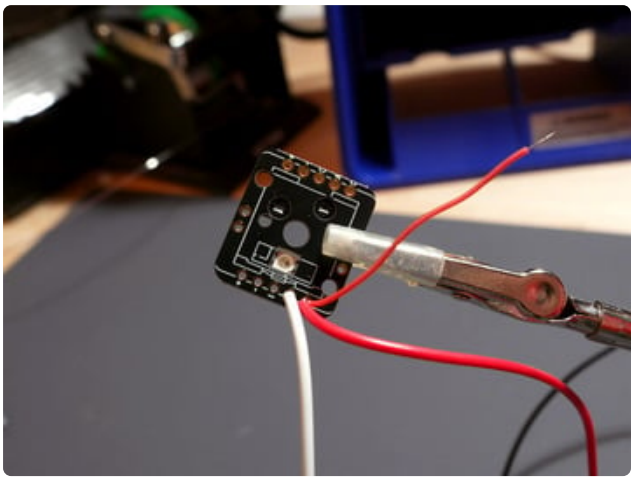


Cut the green wire off of the 4-pin JST-PH cable. Scroll-click won't be used for this project.

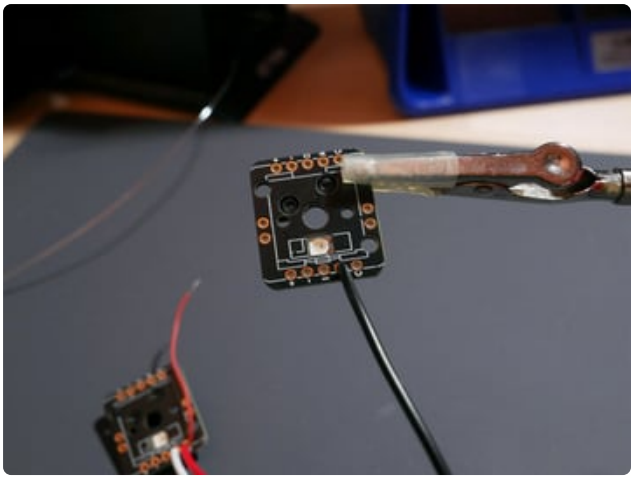
Cut the jumper headers off the wires and splice them.



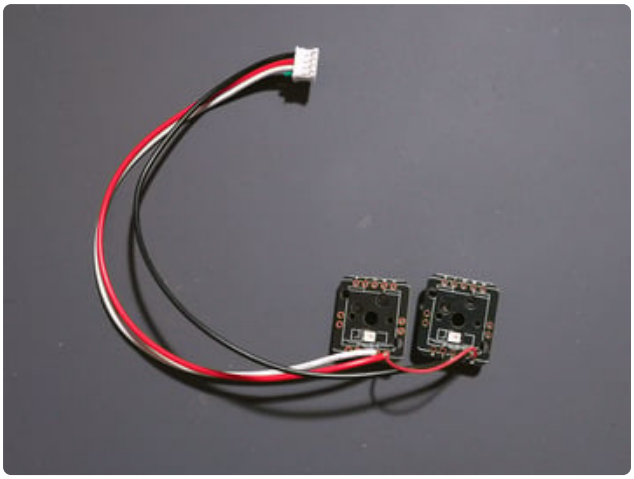
Solder the red wire (GND) to the C pin on a NeoKey. Solder an additional shorter piece of wire to the pin as well.



Solder the white wire (left-click) to the A pin on the NeoKey. This will be the left-click key.

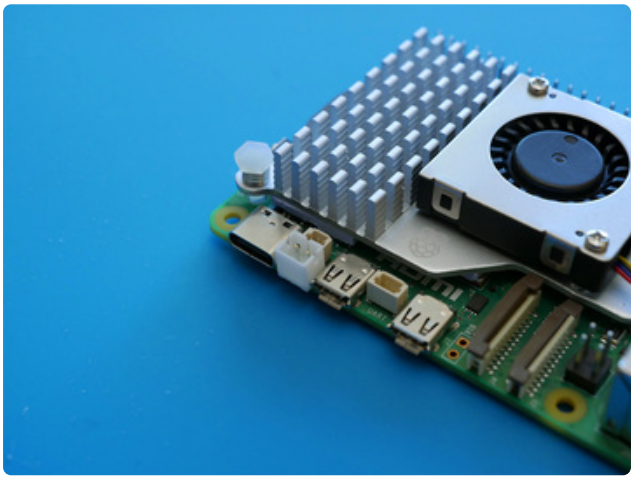


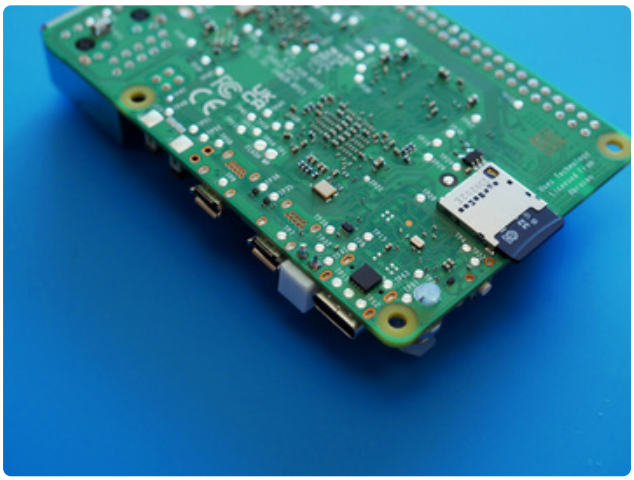
Solder the black wire to the second NeoKey A pin. This will be the right-click key.



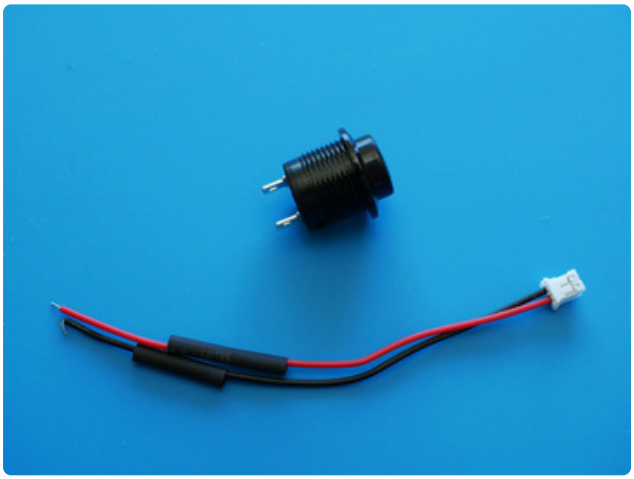
Solder the shorter wire connected to the left-click NeoKey C pin to the right-click NeoKey C pin. This connects GND between the two NeoKeys.

Raspberry Pi Power Button

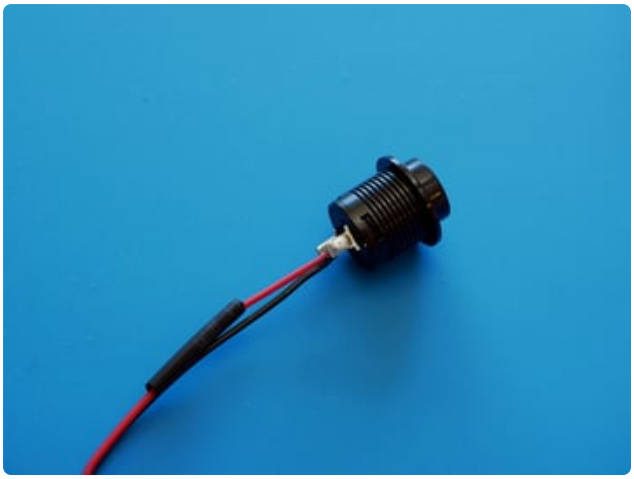




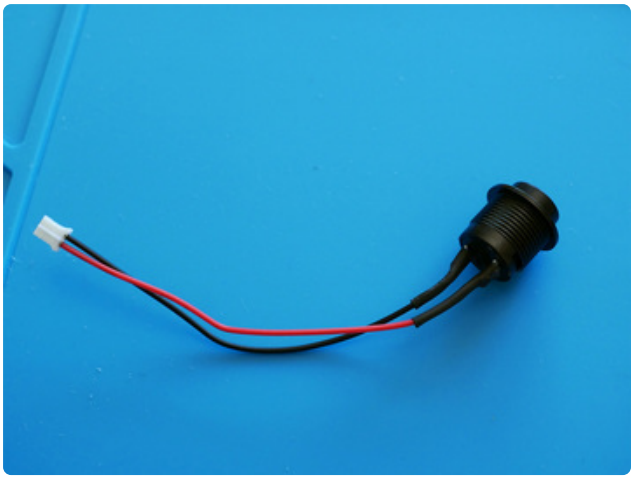
Solder a 2-pin JST-PH socket to the two power button pins on the Raspberry Pi.



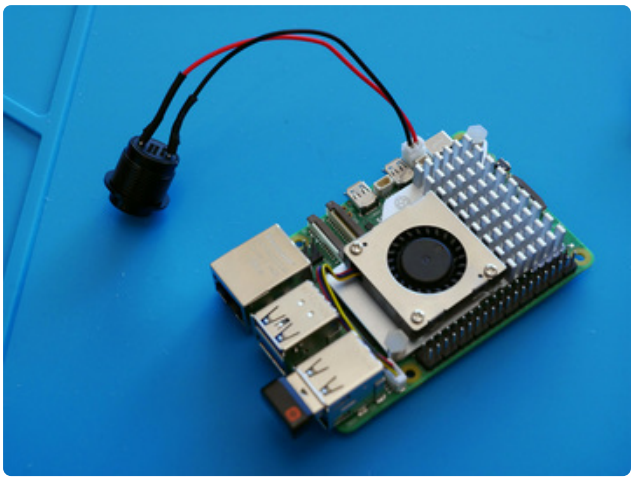
Prep a 2-pin JST-PH cable with heat shrink to solder to a button.



Solder the wires to the button leads.



Cover the leads with the heatshrink.

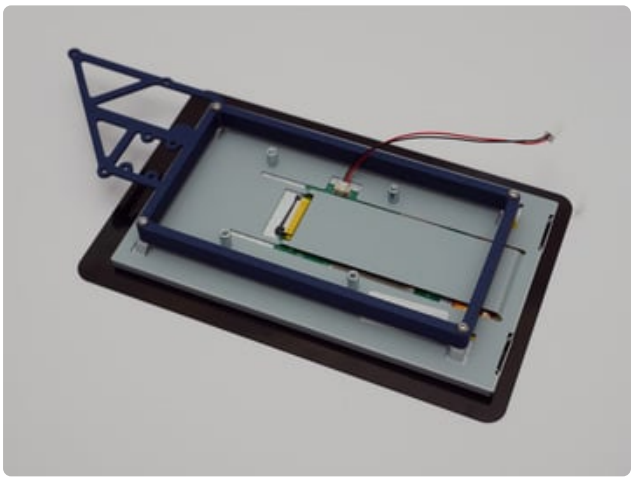


Plug the button into the JST-PH port on the Raspberry Pi.

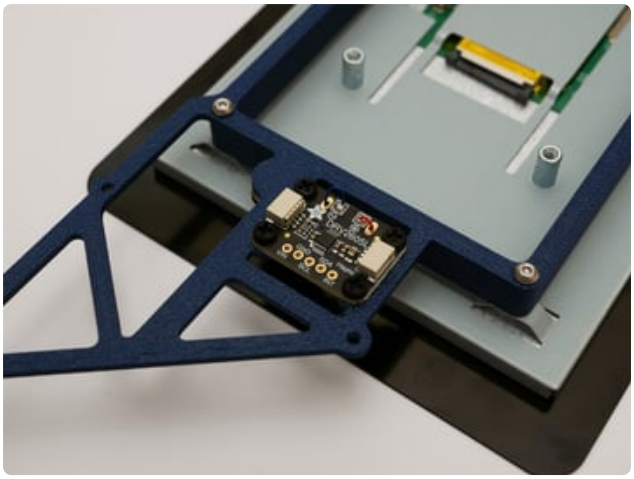
Assembly

Mount the Electronics

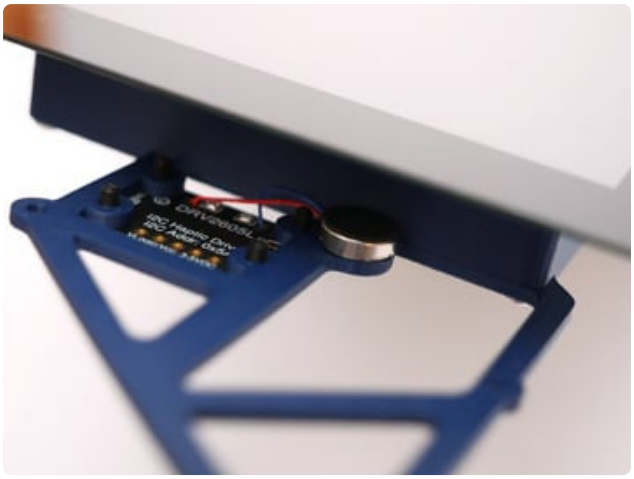
Install the active cooler to the Raspberry Pi 5. Plug in the right angle USB A extender to the Raspberry Pi.



Mount the electronics mounting plate to the back of the display with four 8mm M2.5 screws.



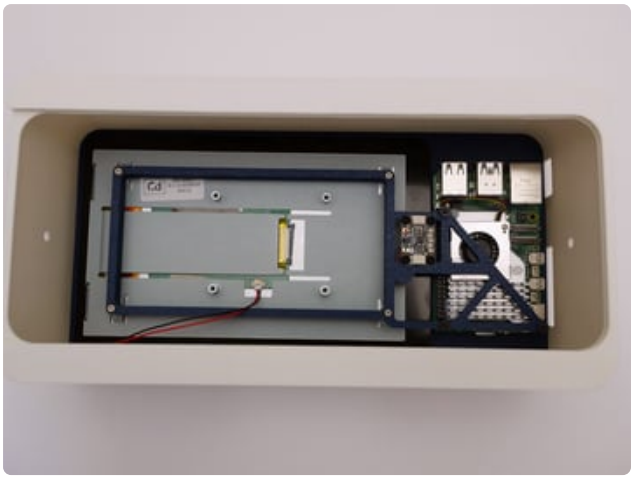
Mount the DRV2605L breakout to the plate with four M2.5 screws and nuts.



Remove the adhesive backing from the vibration motor and stick it to the back circular area of the mounting plate.

Attach three M2.5 standoffs and nuts to the Raspberry Pi section of the mounting plate.

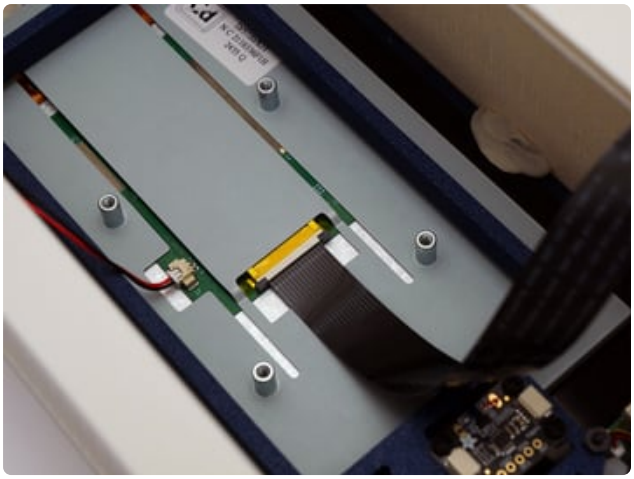
Push the front plate into the main case. Attach four M2.5 screws into the four holes to mount the Raspberry Pi.



Attach the mounting plate to the Raspberry Pi with the M2.5 screws from the front plate to the three M2.5 standoffs.

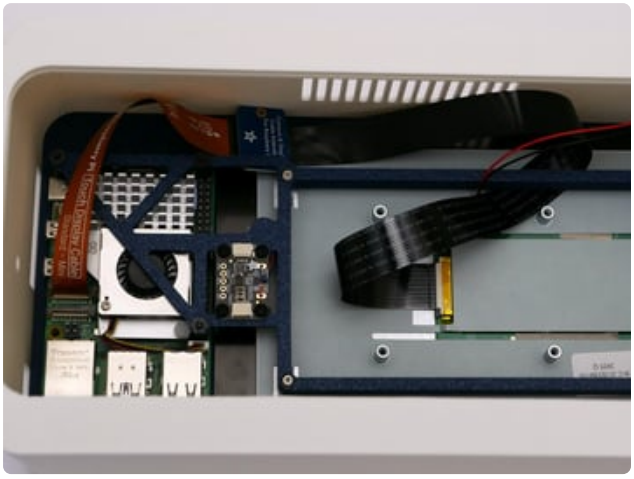
Ribbon Cable

Insert the ribbon cable and the display ribbon cable into the CSI/DSI extender breakout.



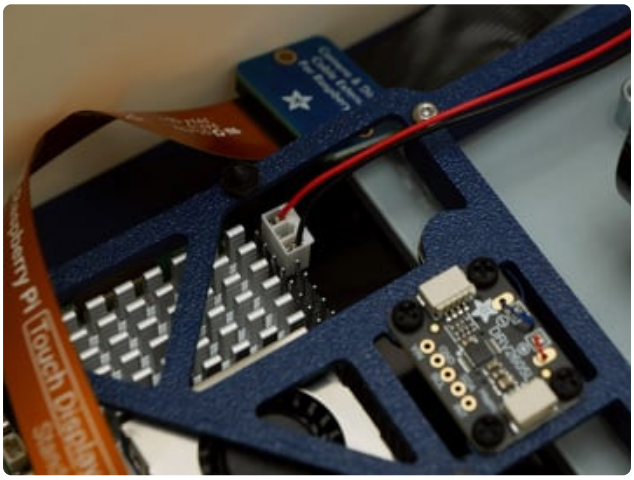
Insert the other end of the ribbon cable into the display.

Insert the other end of the display cable into the port on the Raspberry Pi.

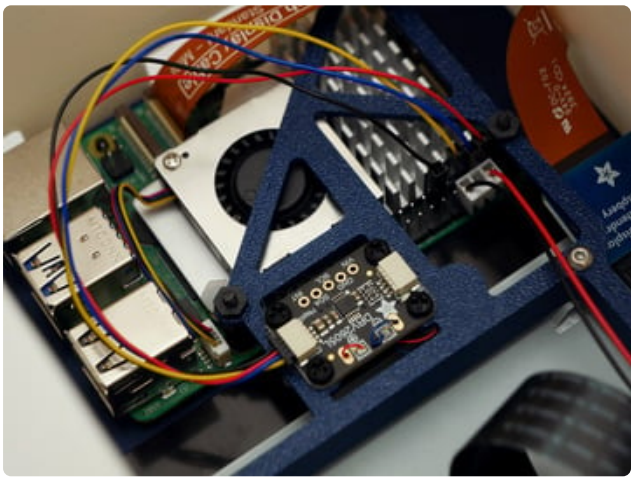


You can cable manage the ribbon cables by sliding the longer ribbon cable underneath the electronics mount.

Pi GPIO



Plug the display power cable into 5V and GND on the Raspberry Pi GPIO header.



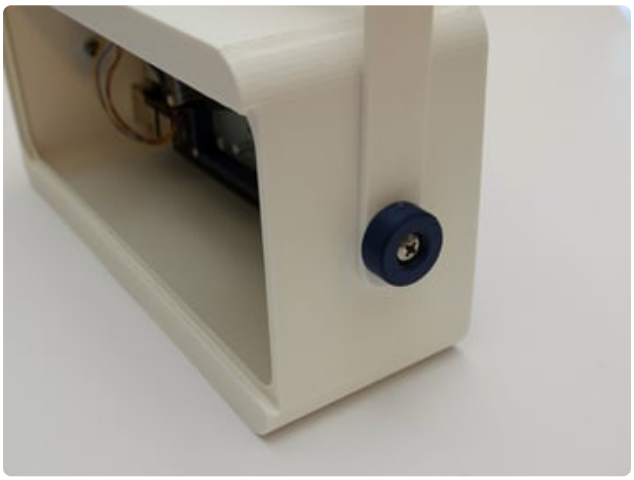
Plug the STEMMA QT cable into the haptic motor breakout. Plug the socket headers into the Raspberry Pi GPIO:

STEMMA 3V to Pi 3.3V (red wire)
STEMMA SDA to Pi SDA (blue wire)
STEMMA SCL to Pi SCL (yellow wire)
STEMMA GND to Pi GND (black wire)

Insert the power button into the button cutout on the side of the case.

Plug in the power button JST PH cable to the JST PH socket on the Raspberry Pi.

Knobs and Acrylic



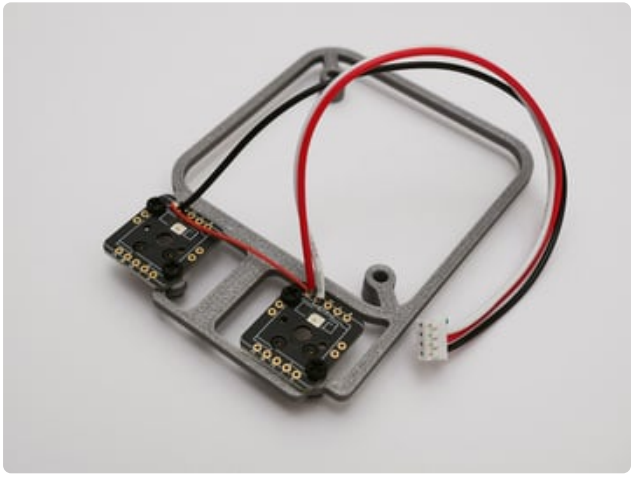
Attach the handle to the case with a knob on each side. Secure them with M4 screws and nuts.

Slide the piece of acrylic into the track in the back of the case.

Place the acrylic pull tab at the edge of the acrylic. When you're happy with the placement, secure it with some glue.

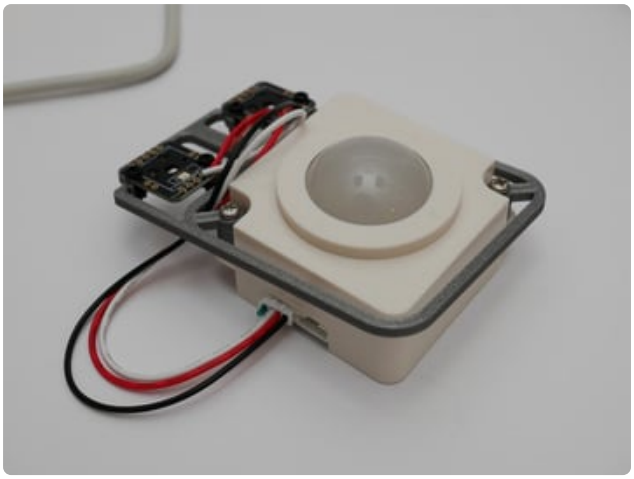
That completes the terminal assembly!

Mouse Assembly

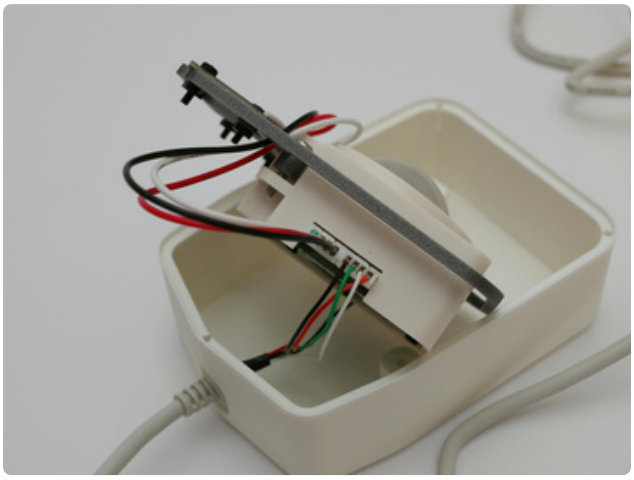


Attach the NeoKey breakouts to the mouse mount with M2.5 screws and nuts.

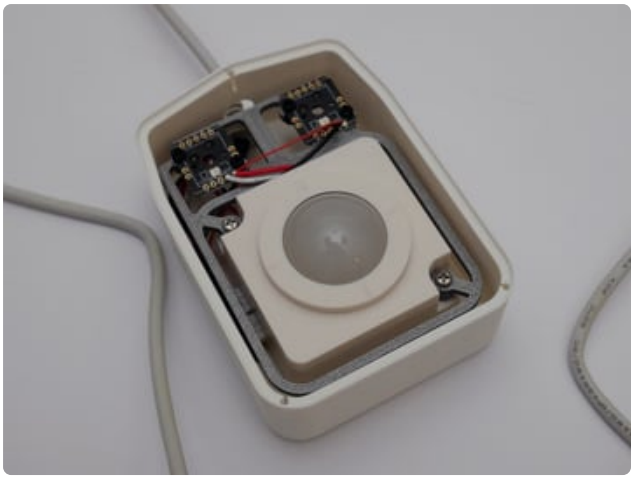
Run the JST end of the mouse USB cable into the mouse enclosure.



Place the mouse mount over the trackball with M3 screws. Plug the NeoKey JST cable into the BUTTON port.



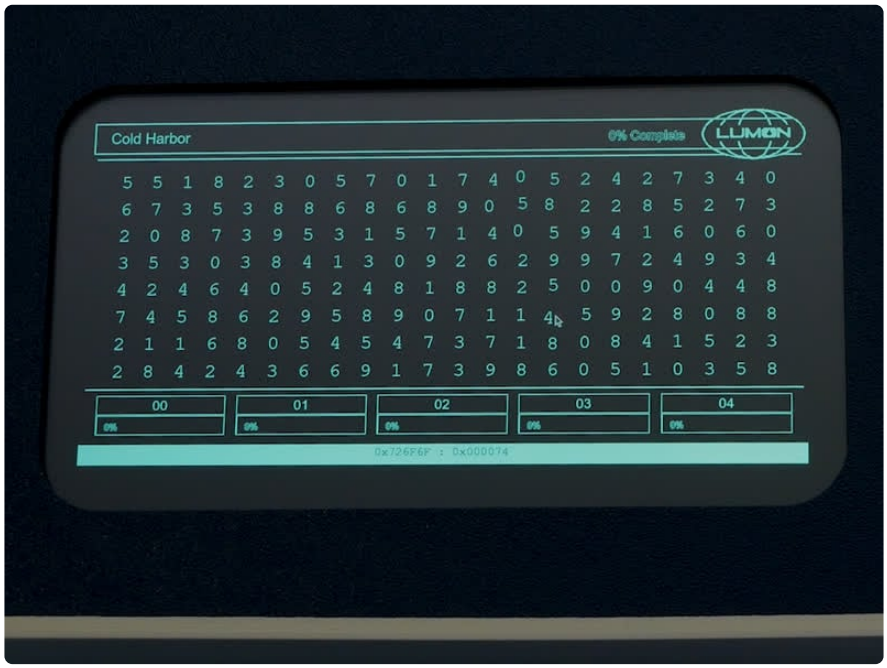
Plug the JST end of the USB cable into the USB port on the trackball.



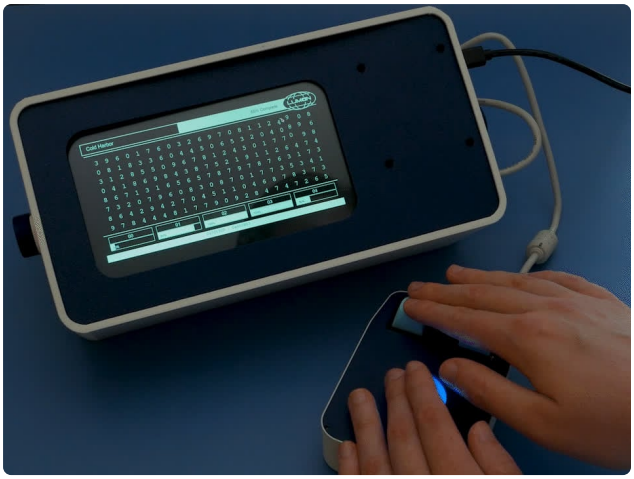
Secure the mouse assembly into the enclosure with the two M3 screws.

Place the front plate into the mouse enclosure. Plug the MX keys with keycaps onto the NeoKey breakouts.

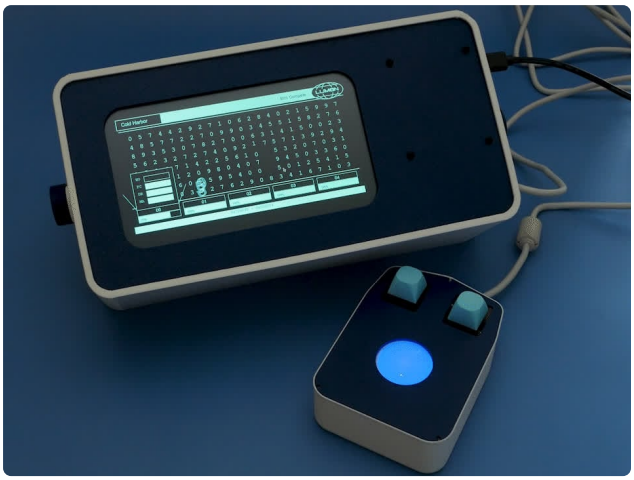
Use



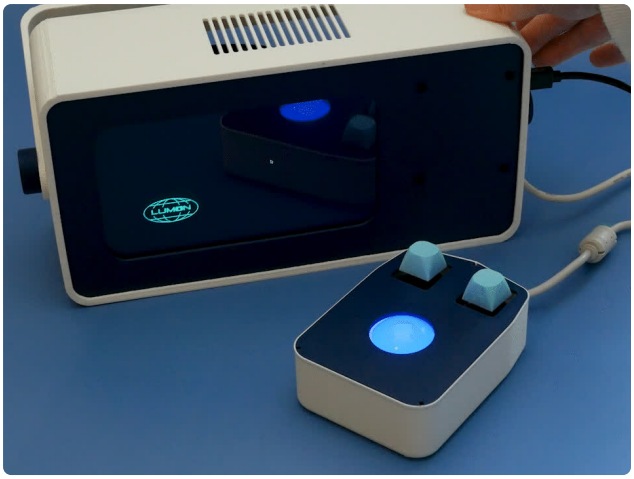
After your Raspberry Pi boots up, the Python program will start running. First you will see the bouncing Lumon logo screensaver. You can toggle between the screensaver and the program with the right-click key on the mouse. Right-clicking also saves your MDR file progress.



Your goal in Macrodata Refinement is to isolate the threatening numbers by drawing a box around them to send them to their respective bins. The threatening numbers will have increased movement on the screen. As your cursor moves closer to them, their movement will become more pronounced and you'll hear and feel an increasing vibration from the haptic motor.



When you complete your file, you'll be greeted with "100% Praise Kier" on the display.



The power button on the side of the chassis lets you safely shutdown the terminal after a long day of refining.