MicroAI™ Atom Raspberry PI APM EVK SDK 1.0

Contents

Disclaimer

Your use of the SDK is at your sole risk. You will be solely responsible for any damage to your computer system or loss of data that results from the download or use of the SDK. To the maximum extent permitted by applicable law, in no event shall ONE Tech or its suppliers be liable for any special, incidental, indirect, or consequential damages whatsoever (including, without limitation damages for loss of business profits or revenue; business interruption or work stoppage; computer failure or malfunction; loss of business information, data or data use; loss of goodwill; or any other pecuniary loss) arising out of the use of or inability to use the SDK or the provision of or failure to provide support services, even if ONE Tech or its supplier has been advised of the possibility of such damages.

What is MicroAI™?

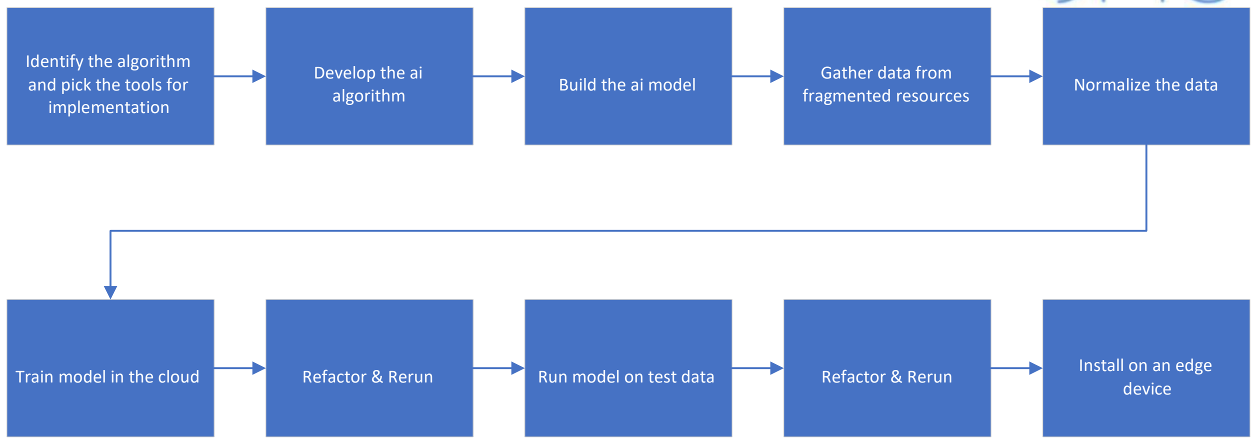
MicroAI™ is an AI engine that can operate on low power edge devices. It can learn the pattern of any and all time series data and can be used to detect anomalies or abnormalities and make one step ahead predictions/forecasts.

Why is MicroAI™ good for your project

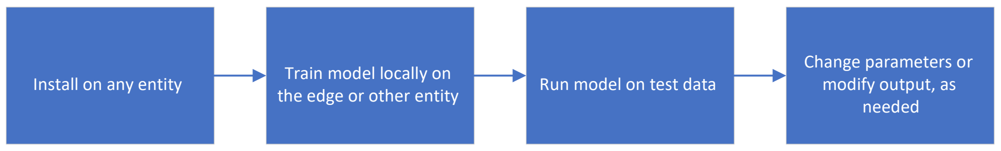
The MicroAI™ SDK enables your raspberry pi device or other entity with our proprietary AI (referred to in this document as 'MicroAI™') to help you build and gather feedback on your use cases, as well as the tutorials covered in this documentation. MicroAI™ can aid in the following tasks:

1. Data Collection
2. Data Organization
3. Model Building
4. Display, edit, and act upon the data gathered
5. Communicate with configured accessories and services to get them to perform actions, like turning on a light or receiving a notification on your phone.

Without MicroAI™ the process for building and integrating your own AI would work as follows:



Using MicroAI™ will save you development and cut down time to production significantly:


Install on any entity
Train model locally on the edge or other entity
Run model on test data
Change parameters or modify output, as needed

Understanding MicroAI™

The MicroAI™ SDK is made up of several key parts. The two most notable are the X-code and Y-code. However, all the parts can be broken down as follows:

Data Channels: Could be sensors, API's, Flat Files, or any avenue to receive time series data.

Data Acquisition(X-code): X-Code is better known as 'input'. One of the foundational pieces of MicroAI is that it is data source agnostic, meaning, that it can ingest data from a variety of sources. Using tools such as Redis and Numpy allows MicroAI to ingest sources such as, ambient temperature, gyroscope, and humidity data with a simple Python script.

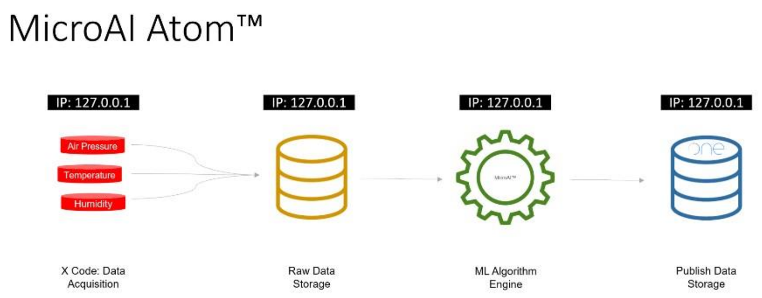
MicroAI: Is made up 3 steps: 1. Training the model 2. Engineering the features 3. Hosting the AI model. However, to define MicroAI simply, it provides insights into device behavior by comparing live data to comprehensive machine learning models.

Output Layer: Once the AI model has been trained and the inputs (X-code) has been configured for the necessary sensors, the MicroAI will begin sending data to the output layer. In this layer, MicroAI can be built up and/or customized to the user's needs.

Application Layer(Y-code): Y-Code is better known as 'output'. However, the application layer and the output layer are dependent on each other. The Application layer provides users flexibility of how they would like to deploy. Essentially, the application layer (Y-Code) can be created to run on top of the output layer for a customized experience. Examples include email alerts and local device behavior changes.

Visualization: This is exactly what it sounds like this can be in the form of dashboards and other outputs.

To visualize the relationship between these entities, please see the following diagram:



While the MicroAI™ libraries are proprietary (and encrypted), open source 'X-Code' (data acquisition) and 'Y-Code' (applications) can be written for multiple solutions.

This document covers MicroAI as implemented in the APM use case. APM is defined below:

APM: APM stands for asset performance management, but, to put it simply, when we refer to APM what we are really saying is that we are attaching external sensors to our raspberry pi device in order to monitor an activity or entity.

Requirements

Supported Devices

MicroAI™ can currently be supported on raspberry pi 3. For APM, you will need some form of outside data from sensors. We recommend using the Raspberry Pi Sense HAT, which can be purchased here: https://www.adafruit.com/product/2738

More devices will be added/recommended over time.

Tools & Versions

Once the MicroAI™ library is embedded in the device, you will need to make sure you have a few tools installed before we get started

- Python 3.7 and Sense Hat library (For APM only)
- Redis-server
- Make sure you have the latest version of Raspbian downloaded from the raspberry pi website (currently Raspbian buster lite)

Preparing your environment

Configuring your pi:

Regardless of if this is your first experience with using a raspberry pi or not, there are some things that need to be done to get you setup for ingesting the SDK.

After booting up your raspberry pi and attaching a monitor and keyboard, enter your username and password. After this, enter the following command:

`sudo raspi-config` then select ENTER

This should open the main menu.

Select number `7 Advanced Options` and select ENTER

Then select `N2 Wi-Fi`

You will need to enter the wi-fi name and the password, then enable.

After enabling Wi-Fi, you will need to reboot your raspberry pi for this change to take effect. Use the following command to reboot after exiting the main menu:

```
sudo reboot
```

Once the raspberry pi has restarted, navigate back to the main menu using the same command as above to enable SSH. This will allow you to remotely access your raspberry pi via the command line or terminal. This will be incredibly helpful as we move into the next steps and you start trying the tutorials.

In the main menu, select number `5 Interfacing Options` and select ENTER

Then select `P2 SSH` and select ENTER. Select ENABLE. After enabling SSH you can exist the main menu and reboot if necessary.

Using SSH:

To after configuring SSH on your raspberry pi, simply open command line or terminal and type in the following command:

`ssh [raspberry pi username]@[IP address of your raspberry pi]` example:

```
ssh pi@123.456.7.890
```

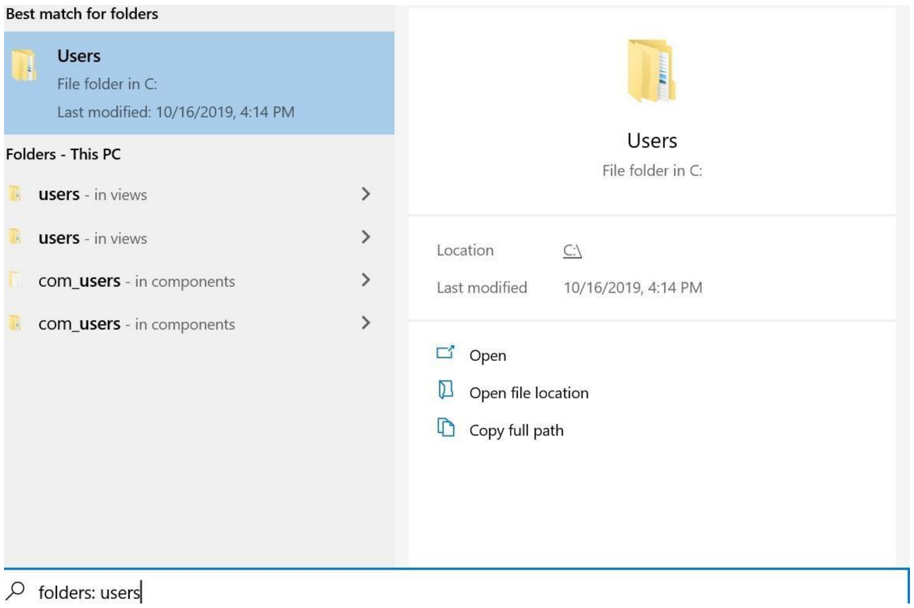
Then simply type in your password, select ENTER and you will have ssh remote access to your raspberry pi.

*To find the IP address of your raspberry pi simply type the following `ip -a` command: *

Possible Error Using SSH on a Windows PC:

If you are a user who has done this before and wanted to start again using a clean raspberry pi, your PC may remember the device (raspberry pi) and not let you access the raspberry pi (it should give an error and a warning of a possible 'man in the middle' attack).

To fix this, search for and open your Users folder.


Best match for folders

Users
File folder in C:
Last modified: 10/16/2019, 4:14 PM

Folders - This PC

users - in views
users - in views
com_users - in components
com_users - in components

Users
File folder in C:

Location C:\
Last modified 10/16/2019, 4:14 PM

Open
Open file location
Copy full path

folders: users

Then select your username > .ssh example: This PC >

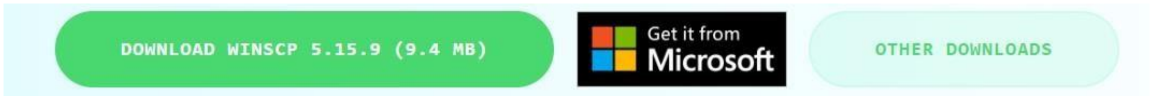
Windows (C:) > Users > john > .ssh

After navigating to your .ssh folder, delete the known_hosts file. This should remove your PC's memory of the device's IP address and you SSH into your raspberry pi again.

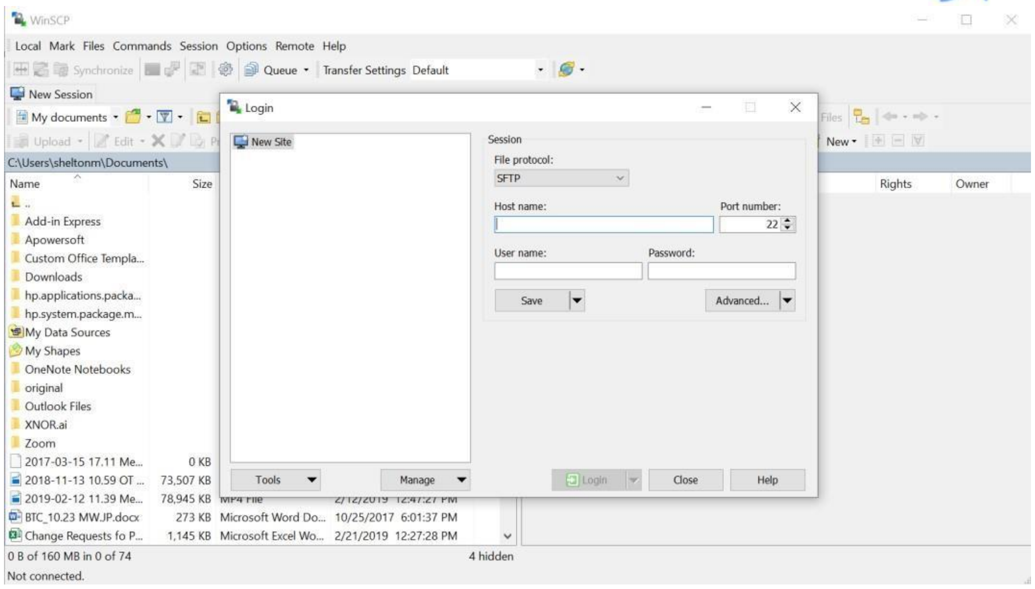
Getting the SDK:

Once your raspberry pi is up and running you will need to do a couple of things to make it a little easier to access and run your use cases. The process is as follows:

1. Go to the ONE Tech URL and download the SDK zip file
2. Extract this file
3. Download the WinSCP file manager here: https://winscp.net/eng/index.php this will help us move the necessary SDK files onto the raspberry pi



4. Once the download is completed open WinSCP and insert the raspberry pi IP address in the Host Name. In the Username and Password, simply type in the username and password for your raspberry pi.



File protocol: SFTP
Host name:
Port number: 22
User name:
Password:

C:\Users\sheltonm\Documents\

```
Name                        Size
..
Add-in Express
Apowersoft
Custom Office Templa...
Downloads
hp.applications.packa...
hp.system.package.m...
My Data Sources
My Shapes
OneNote Notebooks
original
Outlook Files
XNOR.ai
Zoom
2017-03-15 17.11 Me...      0 KB
2018-11-13 10.59 OT ...     73,507 KB
2019-02-12 11.39 Me...      78,945 KB   MP4 File                2/12/2019  12:47:27 PM
BTC_10.23 MW.JP.docx        273 KB      Microsoft Word Do...    10/25/2017  6:01:37 PM
Change Requests fo P...     1,145 KB    Microsoft Excel Wo...   2/21/2019  12:27:28 PM
0 B of 160 MB in 0 of 74                                        4 hidden
Not connected.
```

5. Select Login. Once logged in, drag and newly downloaded and extracted SDK folder from your local computer to the raspberry pi (left to right). After you do this, you will be able to access the SDK files directly on your raspberry pi.

Raspberry Pi Prep:

Note: All the environment prep assets will need to be to be installed on the device (raspberry pi) directly. Any asset being installed on a device other than the raspberry pi (a laptop for instance) will be specifically identified.

The easiest way to get your hands on MicroAI™ and start building is prep your environment properly is by using pip. Be sure that the latest version of pip is installed (pip3), as you will be using this to install all other assets involved in the environment prep.

The first item is Python 3.7 (and above). After installation, we will need to install several python libraries. Now pop open terminal or command line, ssh into your raspberry pi and run the following:

1. sudo apt-get update
2. sudo apt-get upgrade
3. sudo apt-get install python3-pip
4. sudo apt-get install redis-server
5. sudo reboot
6. pip3 install redis

SDK Directory Structure

With the following SDK directory, you will find the following contents:

- **LICENSE_README.txt –** This will cover the legal license agreement you agree to by using MicroAI
- **Overview.txt –** explains at a high level what is in the SDK package

Within SDK Docs

- **Attack_Tool_README.txt –** will explain how to use the attack tool in IoT Security. (Not used in this example)
- **Attack_Tool_Requirements.txt –** will explain the versions of the 3rd party libraries referenced in the attack tool readme
- **microAI_Output_README.txt –** explains class for user created Y-Code

Projects > APM > X

- **SenseHat_signal.py -** begins the data ingestion for MicroAI Network (APM)
- **signalsource_router -** route table for the X-Code

MicroAI Atom Raspberry Pi EVK SDK > Security or APM > AI

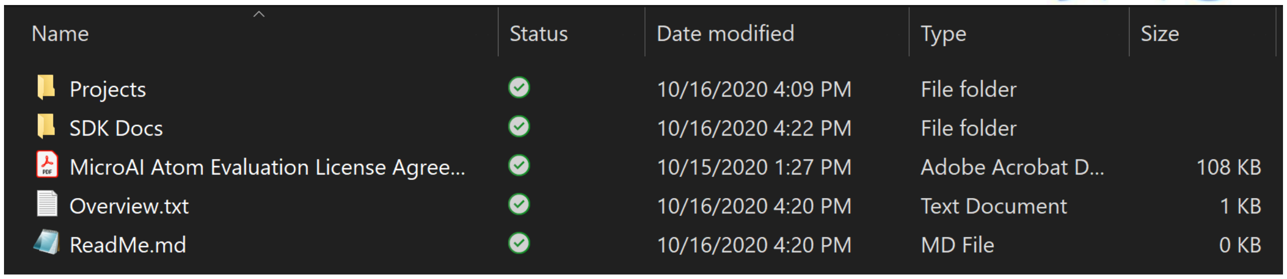
- **AIengine–** executable file that activates the training or execution of the AI generated model
- **AIengine_router–** route table for the AI engine
- **algPar–** Table of parameters that tune the execution of the model
- **rootcfg –** parameters that change the behavior of the training and execution processes
- **signalsource_router –** route table for the AI engine

Projects > APM > Y

- **AIengine_router–** route table for the Y code
- **basic_output.py–** monitors the output of MicroAI and prints if the groups are behaving normally or abnormally
- **MicroAI_Network_Output.cpython-37m-arm-linux-gnueabihf.so** - Encrypted class that allows access to methods for easy use of MicroAI Network's output

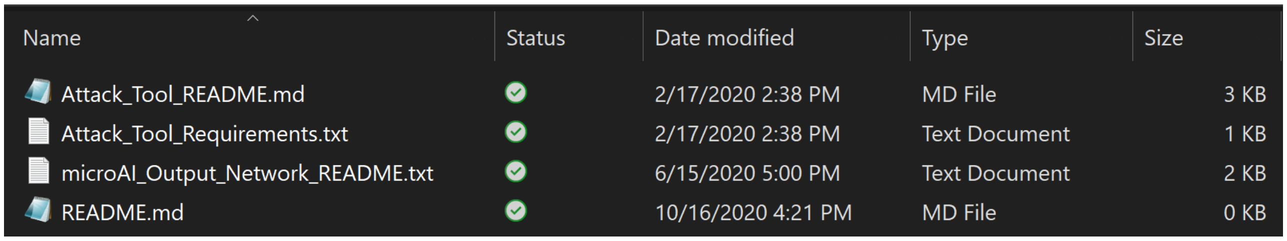
After downloading and extract the SDK zip file, the directory structure should look as follows:

Root:



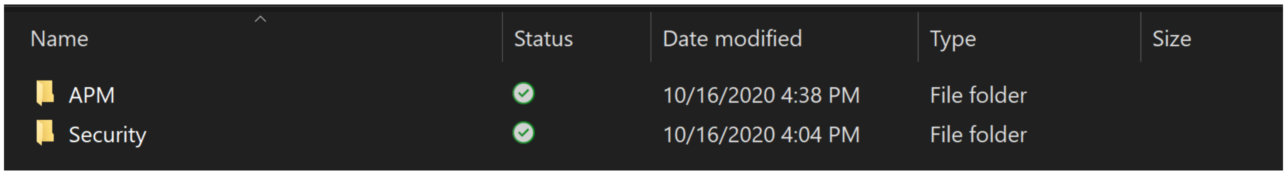
Name	Status	Date modified	Type	Size
Projects	✓	10/16/2020 4:09 PM	File folder	
SDK Docs	✓	10/16/2020 4:22 PM	File folder	
MicroAI Atom Evaluation License Agree...	✓	10/15/2020 1:27 PM	Adobe Acrobat D...	108 KB
Overview.txt	✓	10/16/2020 4:20 PM	Text Document	1 KB
ReadMe.md	✓	10/16/2020 4:20 PM	MD File	0 KB

Root > SDK Docs:



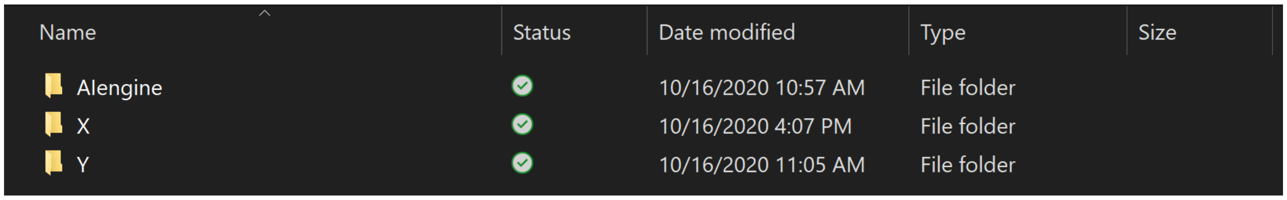
Name	Status	Date modified	Type	Size
Attack_Tool_README.md	✓	2/17/2020 2:38 PM	MD File	3 KB
Attack_Tool_Requirements.txt	✓	2/17/2020 2:38 PM	Text Document	1 KB
microAI_Output_Network_README.txt	✓	6/15/2020 5:00 PM	Text Document	2 KB
README.md	✓	10/16/2020 4:21 PM	MD File	0 KB

Root > Projects



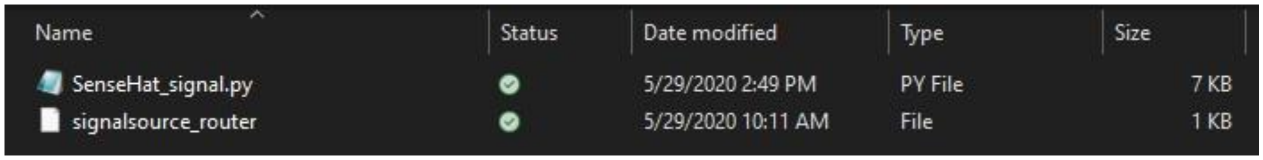
Name	Status	Date modified	Type	Size
APM	✓	10/16/2020 4:38 PM	File folder	
Security	✓	10/16/2020 4:04 PM	File folder	

Root > Projects > APM:



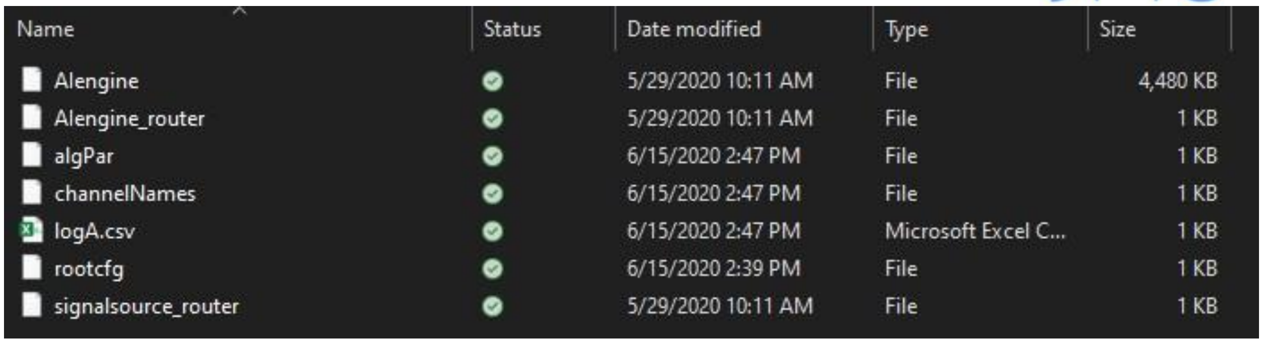
Name	Status	Date modified	Type	Size
AIengine	✓	10/16/2020 10:57 AM	File folder	
X	✓	10/16/2020 4:07 PM	File folder	
Y	✓	10/16/2020 11:05 AM	File folder	

Root > Projects > APM > X



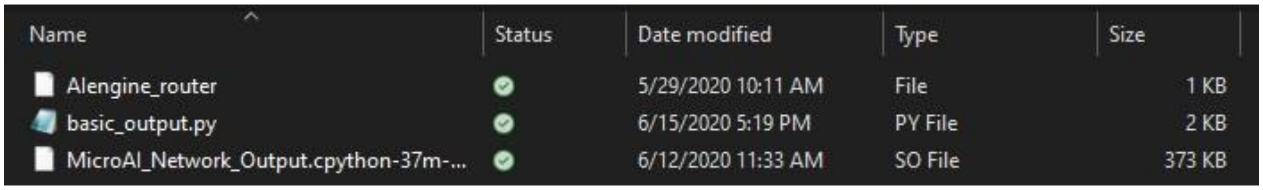
Name	Status	Date modified	Type	Size
SenseHat_signal.py	✓	5/29/2020 2:49 PM	PY File	7 KB
signalsource_router	✓	5/29/2020 10:11 AM	File	1 KB

Root > Projects > Security or APM > AI



Name	Status	Date modified	Type	Size
AIengine	✓	5/29/2020 10:11 AM	File	4,480 KB
AIengine_router	✓	5/29/2020 10:11 AM	File	1 KB
algPar	✓	6/15/2020 2:47 PM	File	1 KB
channelNames	✓	6/15/2020 2:47 PM	File	1 KB
logA.csv	✓	6/15/2020 2:47 PM	Microsoft Excel C...	1 KB
rootcfg	✓	6/15/2020 2:39 PM	File	1 KB
signalsource_router	✓	5/29/2020 10:11 AM	File	1 KB

Root > Projects > APM > Y



Name	Status	Date modified	Type	Size
AIengine_router	✓	5/29/2020 10:11 AM	File	1 KB
basic_output.py	✓	6/15/2020 5:19 PM	PY File	2 KB
MicroAI_Network_Output.cpython-37m-...	✓	6/12/2020 11:33 AM	SO File	373 KB

APM (Using External Sensors)

Configuring your X-Code – APM

(see pg. 4 for definition of APM)

When setting up any APM or raspberry pi use case, the input script and signal source router table must be edited. For the case of this example, these are the only 2 files present in the APM X subdirectory. First let us examine the route table.

```
channelID,IP0,IP1,IP2,IP3,DB1,DB3,groupID
c0,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c1,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c2,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c3,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c4,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c5,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
```

Column 1 is the name of the channel name. This is the name the AI engine will use to identify each channel

Column2 is the IP address of the input signal. The data ingestion code should always check this value to ensure that is either “localhost”, “127.0.0.1” or the local IP of the device that is running the data ingestion or X code. If it is not either of those three values, the X-Code should skip this channel

Colmun3 is the IP address and port number of the redis server where that channel’s input data will be stored

Column4 is the IP address of the MicroAI engine that will be processing the data

Column5 is the IP address and port number of the redis server where the output of the AI engine will be stored

Column6 and 7 are currently unused

Column8 is the group number for the data. This allows the processing of the data to be managed. The output of each channel is partially dependent on all other channels within the same group.

Each row represents a new data channel entry into MicroAI. The above example is configured for 6 entries.

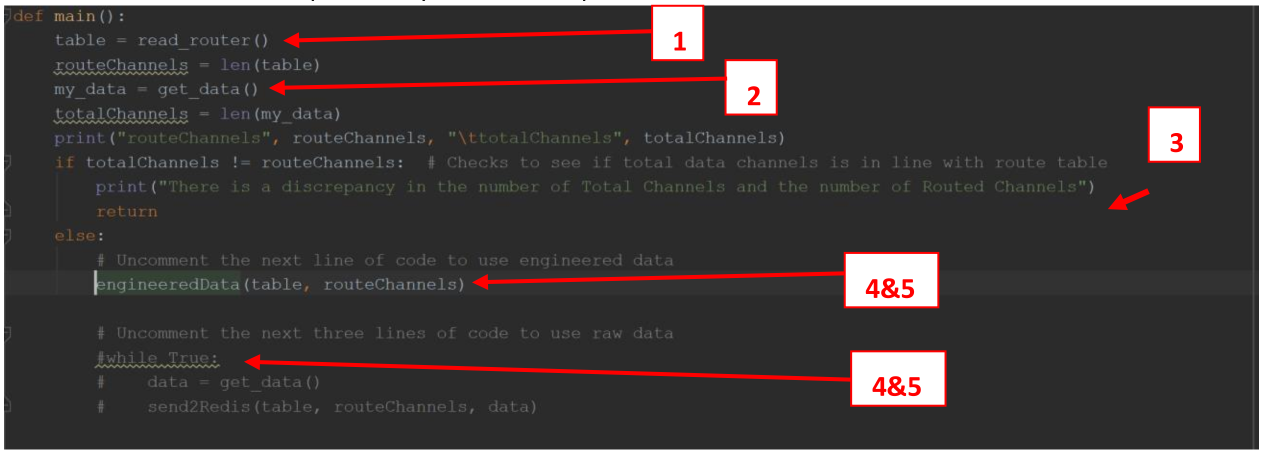
The MicroAI Atom SDK requires that all IP addresses be localhost to function. Other Limitation include:

1. **Max channel numbers = 6**
2. **Max training entries = 3600**
3. **Max execution entries = 129600 (roughly fifteen days of continuous execution)**

In general, the following steps should be taken for all X-Code

1. Read the route table and create a list of all channels that this device will be responsible for creating and where those channels will be stored as well as the channels names(Column 1, Column2, and Column3)
2. Retrieve a sample of raw data
3. Ensure that the sample of Raw data lines up with how many channels the route table states there will be for this device
4. Begin continuously reading data. Engineer your data here if you choose to do so.
5. Set the data channels to the appropriate redis server (limited to localhost for MicroAI Atom) as determined by the route table (Column3). The keys for the redis server should be the channel names assigned in the route table (Column1)

Without going into too much detail, the following code snippet is of the main loop of SenseHat_signal.py and how it fits with the 5 previously outlined steps.

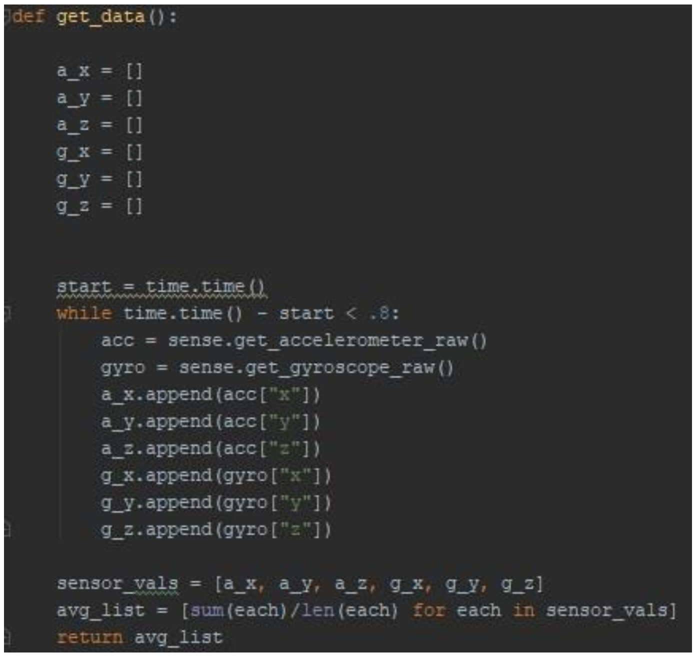


```
def main():
    table = read_router()
    routeChannels = len(table)
    my_data = get_data()
    totalChannels = len(my_data)
    print("routeChannels", routeChannels, "\ttotalChannels", totalChannels)
    if totalChannels != routeChannels:  # Checks to see if total data channels is in line with route table
        print("There is a discrepancy in the number of Total Channels and the number of Routed Channels")
        return
    else:
        # Uncomment the next line of code to use engineered data
        engineeredData(table, routeChannels)

        # Uncomment the next three lines of code to use raw data
        #while True:
        #    data = get_data()
        #    send2Redis(table, routeChannels, data)
```

For the example, get_data() returns the x,y, and z components of both the gyroscope and accelerometer sensors on the board. In other words, it returns 6 channels which is in line with 6 channels outlined in signalsource_router. From there, the user can either use engineered data or raw data by uncommenting either of the blocks of code within the else statement. By default, engineered data is selected.

An example of get_data() is shown below.


```
def get_data():

    a_x = []
    a_y = []
    a_z = []
    g_x = []
    g_y = []
    g_z = []


    start = time.time()
    while time.time() - start < .8:
        acc = sense.get_accelerometer_raw()
        gyro = sense.get_gyroscope_raw()
        a_x.append(acc["x"])
        a_y.append(acc["y"])
        a_z.append(acc["z"])
        g_x.append(gyro["x"])
        g_y.append(gyro["y"])
        g_z.append(gyro["z"])

    sensor_vals = [a_x, a_y, a_z, g_x, g_y, g_z]
    avg_list = [sum(each)/len(each) for each in sensor_vals]
    return avg_list
```

How to use it

Engineering features – APM

(see pg. 4 for definition of APM)

For some use cases, the raw data of a sensor is good enough, but for others it does not tell the full story. These exceptions require 'Feature Engineering' to transform the raw data into usable data.

For example, say a machine needs to run at 70 degrees Celsius. It is known that over the lifetime of the machine, the operational temperature will rise to some unknown higher temperature. In this scenario, if the lifetime of the machine is long and the rise in temperature is slow, MicroAI™ would not be able to detect the rise in temperature as being abnormal. In this case, it will constantly readjust its definition of what is 'normal' until the machine breaks down due to sustained use at the higher temperature.

Do not fret! This problem can be solved with some handy Feature Engineering. For this example, we could round the temperature from the sensor down to the nearest whole number and subtract 70 from it. This means, in the beginning, the MicroAI™ would always be fed a 0. Eventually, the temperature would rise high enough to be a 1.

That dramatic shift from 0 to 1 would trigger an abnormal behavior flag in the MicroAI™ and the users would know that it is beginning to show signs of breakdown. This pattern would then repeat for every degree increase in temperature. MicroAI™ should detect abnormal behavior every time the temperature increases by 1 degree.

Another example of feature engineering would be if you want to measure vibration of some entity, but, you only have positional sensors. The positional sensors will return an x, y, and z values, but, those values alone will obviously not be able to tell us the 'vibration' that the sensor is experiencing. Rather, we could assume that, vibration (in this case), would be the change or difference from one position to the next (values of x, y, and z).

To clarify, the positions would have to be compared to their previous position to determine vibration. This can be done by taking the absolute value of the difference between the previous value of the sensor and the current value. With this new value, we now we have a usable measurement for vibration and MicroAI™ can begin learning its normal behavior.

It is important to note that all feature engineering should occur in the X-Code. Instead of passing the raw data into the return statement, the transformed or modified data should be passed. If the computation for transforming the data is longer than the samplingTime set in rootcfg (in the AIengine directory), then samplingTime must be increased, or the transformation must be modified so that it can occur more quickly.

Training & Running the Model

Before the MicroAI™ engine can be run, it must first construct a training model. The more data points the training model has, the more accurate it should be. It should be noted that having more data points does not increase the size of the model but does increase the accuracy. First let us leave the X-Code directory and enter the AIengine directory

```
cd..
```

```
cd AI/
```

To begin training the model, open rootcfg.

```
nano rootcfg
```

First ensure that 5ifBuildModel is 1 and 6ifExecuteModel is 0. Next we will edit 3hDim and 4tDim. 4tDim is the number of data points that will be used to train the model. By default, it should be 200. 3hDim should be set to be 1/10th the size of 4tDim. So, for our case, we shall set it to 20. Save the file and exit back to the terminal.

Next we will train the model by running

```
./AIengine
```

We will now wait for the AI to finish training. This should take approximately 4tDim * 16samplingTime milliseconds. So, in our case, it should take around 400 seconds or roughly five and a half minutes. Once this process is complete, we need to edit rootcfg again

```
nano rootcfg
```

This time we will set 5ifBuildModel to 0 and 6ifExecuteModel to 1. The next time we activate the AIengine executable, it will not train the model, but instead activate the AI.

```
./AIengine
```

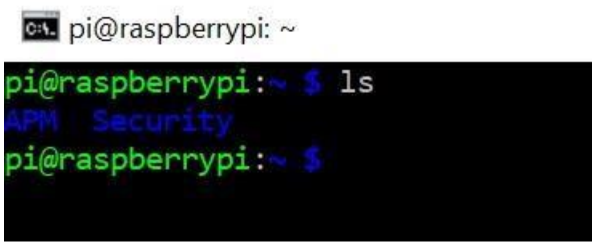
APM Tutorial (Movement Tracking) – Step 1

(see pg. 4 for definition of APM)

By default, these tutorials will be preloaded in the SDK. To begin, open command prompt, ssh into your raspberry pi, and type the following command:

```
ls
```

You should see two directories: APM and Security



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ ls
APM  Security
pi@raspberrypi:~ $
```

You should then cd into the directory/tutorial you want to run through by typing the following command:

```
cd directory_name
```

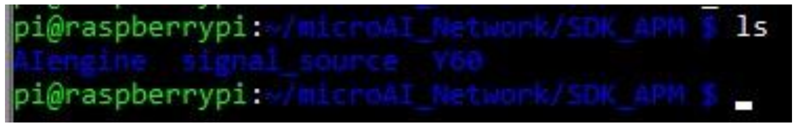
After 'cd' type the name of the directory (APM)

Step 2

Next let us view the files and directories that are present.

```
ls
```

You should see the following file structure:



```
pi@raspberrypi:~/microAI_Network/SDK_APM $ ls
AIengine  signal_source  Y60
pi@raspberrypi:~/microAI_Network/SDK_APM $ _
```

From here we will want to open a tmux window:

```
tmux new -s APM
```

tmux will allow us to open multiple command prompt sessions within the same window. To run MicroAI, we will need at least 3 windows open. To open additional windows in tmux type CTRL+B, " to split your current window vertically and CTRL+B, % to split your current window horizontally. To switch between windows, use CTRL+B, "arrow key".

First let us start the X-Code.

```
cd signal_source
```

Let us check the signalsource_router file to ensure that it looks like the following

```
channelID,IP0,IP1,IP2,IP3,DB1,DB3,groupID
c0,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c1,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
```

```
c2,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c3,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c4,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
c5,127.0.0.1,127.0.0.1:6379,127.0.0.1,127.0.0.1:6379,0,0,1
```

```
nano signalsource_router
```

Once you have confirmed the route table, save, and exit from the file. Now we will activate the data ingestion or X-Code

```
python3 SenseHat_signal.py
```

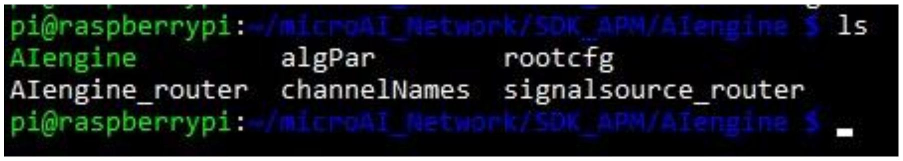
Training & Running the Model

We now need to train and activate the AIengine. Switch to a different tab in your tmux window and then enter the AIengine directory

```
cd AIengine
```

First let us look at the files present within this directory

```
ls
```



```
pi@raspberrypi:~/microAI_Network/SDK_APM/AIengine $ ls
AIengine          algPar        rootcfg
AIengine_router   channelNames  signalsource_router
pi@raspberrypi:~/microAI_Network/SDK_APM/AIengine $
```

The first step is to ensure that both the AIengine_router file and the signalsource_router file are identical to the signalsource_router file in the signal_source directory. Once this has been accomplished, we must now configure some settings.

```
nano algPar
```

```
0.8,0.8,3,1,0,0,0
0.8,0.8,3,1,0,0,0
0.8,0.8,3,1,0,0,0
0.8,0.8,3,1,0,0,0
0.8,0.8,3,1,0,0,0
0.8,0.8,3,1,0,0,0
```

Ensure that algPar has as many rows as there are channels being processed by the AIengine. For this case, algPar should have 6 rows. The first column should have values between 0 and 1. Increasing the number will lower sensitivity (less behavior will be considered abnormal) The second column should also be a number from 0 to 1, but increasing this number will increase sensitivity. The third column should be a positive number greater than 0. Increasing this number will lower sensitivity. The remaining columns should not be changed. Save and exit the file with CTRL-S, CTRL-X.

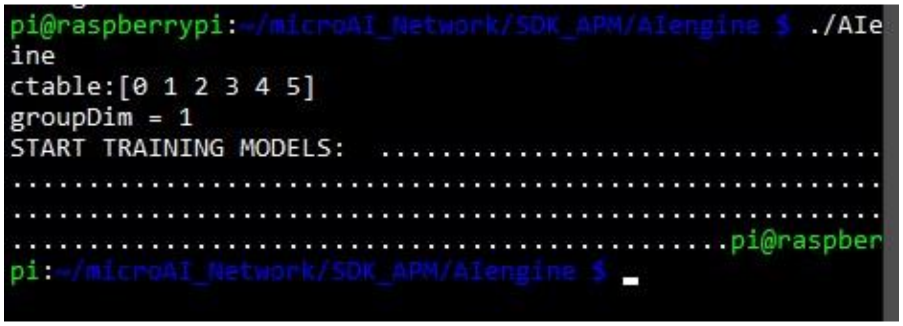
Next we will open the next configuration file.

```
nano rootcfg
```

set 5ifbuildModel to 1 and 6ifexecuteModel to 0 and then save and exit. Now we will begin the AI
training process

```
./AIengine
```

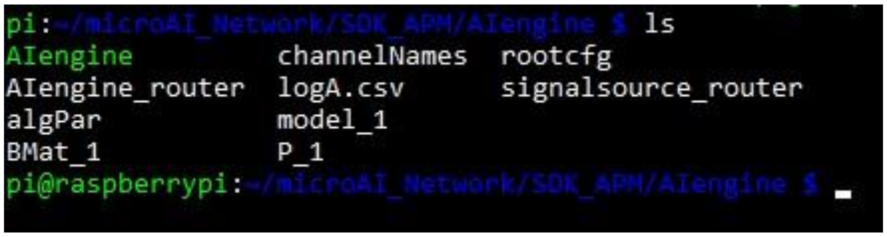
Once the training process is complete, you will see the following message



```
pi@raspberrypi:~/microAI_Network/SDK_APM/AIengine $ ./AIe
ine
ctable:[0 1 2 3 4 5]
groupDim = 1
START TRAINING MODELS:  ................................
.........................................................
.........................................................
............................................pi@raspber
pi:~/microAI_Network/SDK_APM/AIengine $ 
```

Now let us check the files present within the directory. We should see three new files: Bmat_1, P_1, and model_1.

```
ls
```



```
pi:~/microAI_Network/SDK_APM/AIengine $ ls
AIengine          channelNames  rootcfg
AIengine_router   logA.csv      signalsource_router
algPar            model_1
BMat_1            P_1
pi@raspberrypi:~/microAI_Network/SDK_APM/AIengine $ 
```

Now we will configure the executable to activate the engine.

```
nano rootcfg
```

Set 5ifbuildModel to 0 and 6ifexecuteModel to 1 and then save and exit.

```
./AIengine
```

The engine is now up and running. Switch to the last tmux tab. Let us run a simple application script or Y-Code to see its output. First Navigate to the Y-Code directory

```
cd Y60
```

From here you must ensure that the route table in this directory is identical to the ones in the previously discussed directories.

```
nano AIengine_router
```

After you have made any necessary changes to the route table, it is time to activate the Y-Code.

```
python3 basic_output.py
```

This script will loop forever as it prints the behavior of each group within the MicroAI every 2 seconds.

Limitations and Dependencies

- Currently, the only accepted device for running MicroAI is the raspberry pi 3. However, you can also run MicroAI on a virtual machine(VM) in some use cases. Before running MicroAI on a VM, the user should note that fairly this use case is for advanced users experienced in trouble shooting both database and other programming related issues.
- Data will not be directly stored.
- Future versions of packages and devices may not be supported. Additional documentation will be given to accommodate updates.
- This is an evaluation version only and should not be expected to support a full production environment.

- **Max channel numbers = 6**
- **Max training entries = 3600**
- **Max execution entries = 129600 (roughly fifteen days of continuous execution)**

Reference Materials

- For questions or to get help please post your questions here: http://developers.ONE Tech.ai/community/

FAQ

Can I use other sensors on my raspberry pi 3?
Yes, other sensors are accepted, however, the one documented here is a sense HAT.

Can I modify MicroAI™?
No. You may only modify X-code and Y-code. Which is essentially the input and output.

Is there going to be a GUI for the MicroAI™ configuration, or will it always be command line/terminal based?
MicroAI™ is currently for use by developers in command line/terminal.

Is MicroAI™ supervised or unsupervised learning?
MicroAI™ uses semi-supervised learning. Model training does occur; however, it also learns on its own, in real time.

Would it be possible to change the metrics of certain values, so they better meet my needs, such as Celsius to Fahrenheit?
Yes, on the application layer. Users can convert values using different units.

After you download the MicroAI™ SDK on your computer, do you need to run any installation?
No, once you have the file unzipped and your python dependencies installed, you can get started with MicroAI™.

If connecting to the device using SSH, do users need a VPN into a certain network for MicroAI™ to work?
No, the machine they are SSHing from just needs to be on the same network as the device they are running MicroAI™ on.

How far ahead of time can MicroAI™ forecast?
MicroAI™ predicts values one step ahead of the current value. This could be one second ahead or hours ahead based on the output frequency.

Will MicroAI™ be able to detect anomalies that happen over long periods of time?
This will depend on how long the AI model is trained, as well as, the frequency of change.

For training, can MicroAI™ intake data that is not part of real time asset data? Such as industry standard levels over time or lifetime data of a similar asset?
Yes, MicroAI™ can intake historical data as part of the training dataset.

Can different parameters fed into MicroAI™ be given different weights?
Yes, using feature engineering capabilities, MicroAI™ can incorporate different weights of channel values for optimal predictive analysis.

What is the limit of devices or channels that can be on a single MicroAI™ deployment?
Currently, MicroAI™ SDK supports 6 channels being ingested from a single edge device. However, full versions of the service can support hundreds of channels

Permission Denied. How do I get execution rights to an executable file?

Some users will experience a permission denied error when attempting to run the executable files in the demos. To remedy this, use the command

```
chmod +x executable_name_here
```
.