Tiny Cow: Smart Weather Prototype

Introduction

For the past couple of years, StreamLogic has been actively developing various tools for and proof-of-concept applications of tinyML for low-power, low-cost hardware. The primary focus has been on image and audio use cases. The AI for Good TinyML Challenge was a great opportunity to use that experience in an effort to do something that was not only interesting and challenging, but also something with the potential to have real impact for good in our world.

After a great deal of brainstorming, we chose to focus on measuring the amount of accumulated rainfall at high resolution. Although I believe there are some potential methods to measure wind using audio data, the development would have been substantially longer and it was not possible to work on both rainfall and wind in the given timeframe. Our first approach to measure rainfall, which used image data for rainfall prediction, did not work out (more information about that at the end of this report). So, in the end, this work was focused on high-resolution accumulated rainfall prediction from audio data.

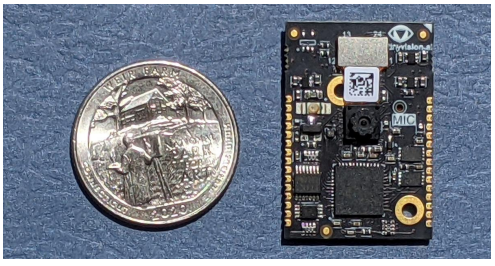
In addition to the rainfall prediction model, a great deal of effort was spent on optimizing power for deployments with battery power. This resulted in two more predictive models used to help extend battery life. The following sections describe our work to develop the rainfall model as well as the two models used to minimize power consumption.

Rainfall Predictive Model

The goal of this model is to collect some seconds of audio data, predict the current rainfall intensity, and then post-process the predicted intensities over time to output sub-hourly rainfall accumulation. Intensity prediction is output by a neural network model and custom code is used to post-process that data into the predicted amount of accumulation.

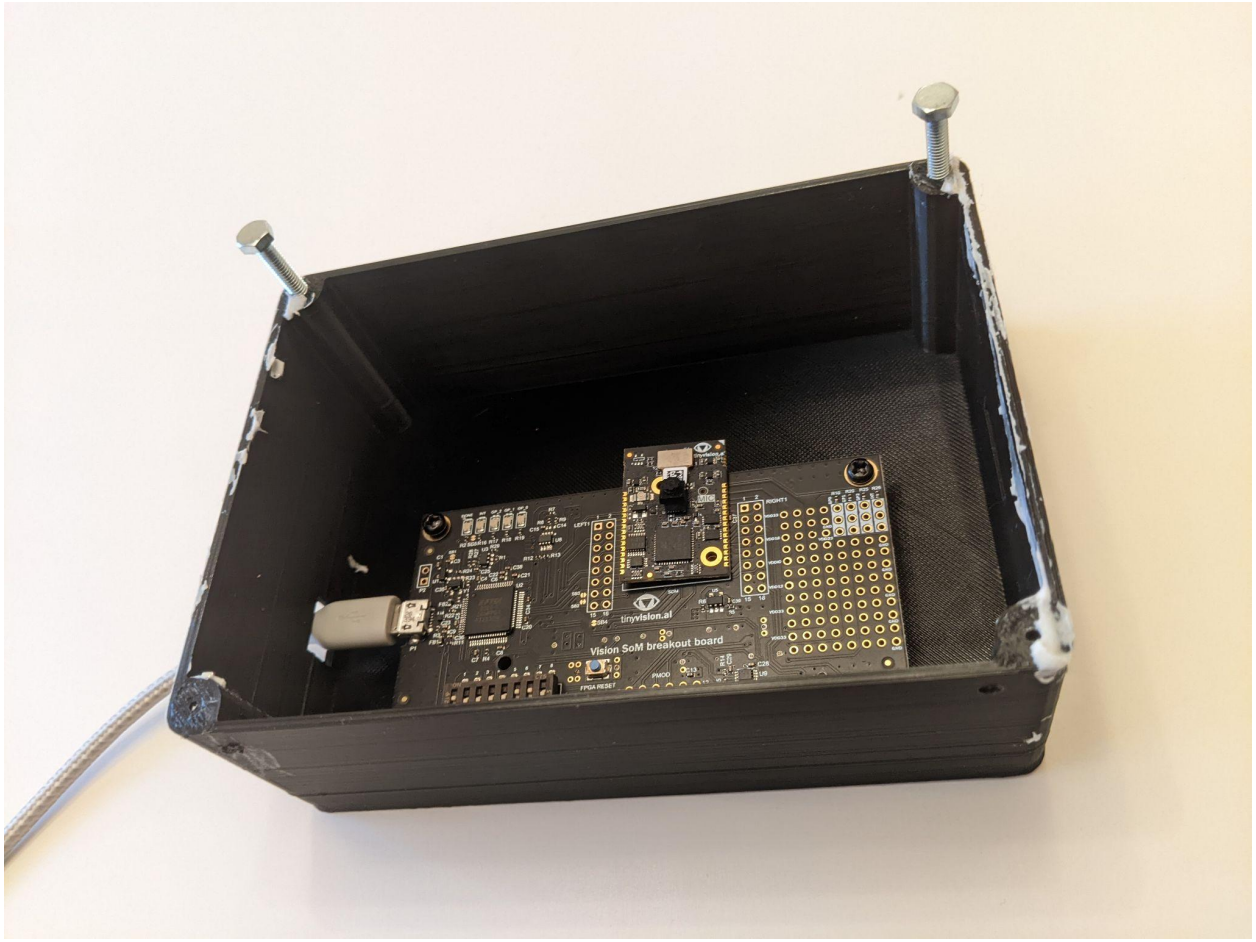
Hardware

The “Vision FPGA SoM” from tinyvision.ai was used for audio data collection and processing.



This tiny board includes a low-power image sensor, microphone and a low-power FPGA for processing. The camera module is removable and not used in this project.

Although this board can be deployed stand-alone, for the prototype, it is mounted on a development board inside a 3D printed project box.



To help produce a stronger audio signal and improve the detection of light rain, a funnel is used over the box as a “concentrator.” This allows smaller drops to combine into larger ones before hitting the measurement box.

The box also has an angled surface, so that water does not pool on top which might dampen the sound of falling drops. Different types and sizes of concentrators may be better depending on the type of expected rain which varies regionally. Further experimentation is needed here.

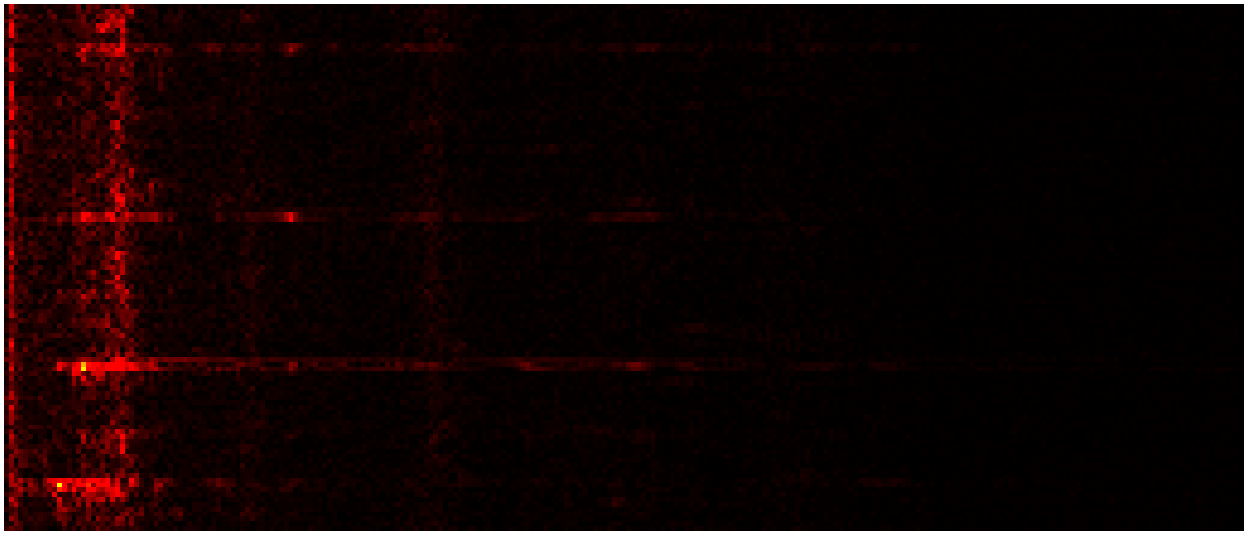
Preprocessing

Audio data can be input to neural networks in a number of ways including raw signal data from the microphone sensor, spectrogram data, or MFCC features. MFCC is primarily useful for speech and will not be described here. Raw data or spectrogram would be suitable choices for

this application. However, we only have experience training models on spectrogram data, so that was used again for this application.

A spectrogram is a 2D array of data constructed from raw audio samples over a predetermined time window. Each row of the spectrogram is the output of the Fast Fourier Transform (FFT) applied to some audio samples to convert the audio into the frequency domain. Several FFTs are computed over time and concatenated to form the 2D spectrogram.

Visual inspection of sample spectrogram images shows that drops landing on the project box produce distinct spikes across a large range of frequencies (i.e. horizontally in the image).



Data collection

Obtaining data for training was challenging. Often when working with image tasks or speech tasks, one can find large public datasets to at least pre-train with. In this case, there is no public data to rely on. Collecting your own data has its own challenges. First, there were not a lot of rainy days in our region during the challenge timeframe. Second, as our approach evolved, we had to collect new data. Finally, it was difficult to obtain ground truth to accompany the collected audio data.

At first, a nearby weather station from the Weather Underground was used as ground truth. There were a number of problems with that approach:

- The measured rain did not always correspond with our exact location.
- The weather station did not pick up light rain.
- The tipping bucket instrument does not capture high-resolution variation in rain intensity.

After some time, we discovered a smart scale that was able to measure and log the weight of collected rain at configurable intervals. Our final results are based on spectrogram data

collected from the prototype together with weight measurements from the scale. Unfortunately, we were only to capture three days of data before the challenge's end date.

The collected data includes many spectrogram's captured and stored in image format and an index of the files together with the corresponding weight measured stored in csv format. The complete dataset may be downloaded from the following URL:

https://1drv.ms/u/s!AjkYZezDnHDlgb1AfeJHF9drG2tBRg?e=7kDxcz

Neural Network

A regression neural network was designed and trained using Tensorflow. The network accepts the spectrogram as input and predicts the amount of rainfall collected during the timeframe of the spectrogram (10 sec) in grams. Converting grams to standard measures such as inches per hour is straight forward.

A great deal of time was spent exploring various neural network architectures and parameters. These are some of the lessons learned from this exploration:

- Special care needs to be taken to deal with the unbalanced nature of the data. Our approach was to bin the samples into ranges and oversample underrepresented bins.
- Square 2D convolutions are not as effective as reducing first horizontally and then reducing vertically.
- Training well past the point at which the loss function levels out significantly improved the model. This suggests that although the mean squared logarithmic error loss was effective at training the model, it was not a good indicator of actual end-to-end performance.

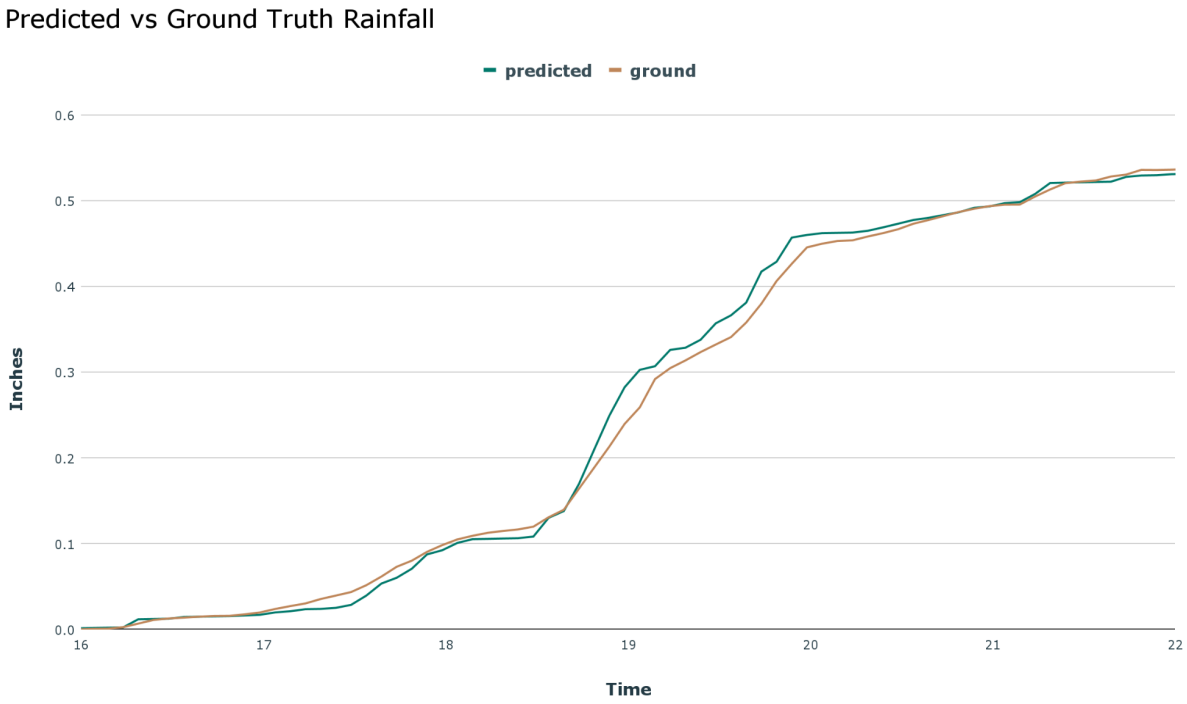
The final model had only 2800 parameters and is easy to implement on small devices. The models layers are:

Layer	Kernel	Stride	Outputs
Conv2D	5x5	2,2	5
BatchNorm			
Conv2D	5x5	2,2	5
BatchNorm			
Dropout(30%)			
Conv2D	1x32	1,1	5
Conv2D	9x1	1,1	8
Dropout(30%)			
Conv2D	9x1	5,1	1
GlobalAverage			

Results

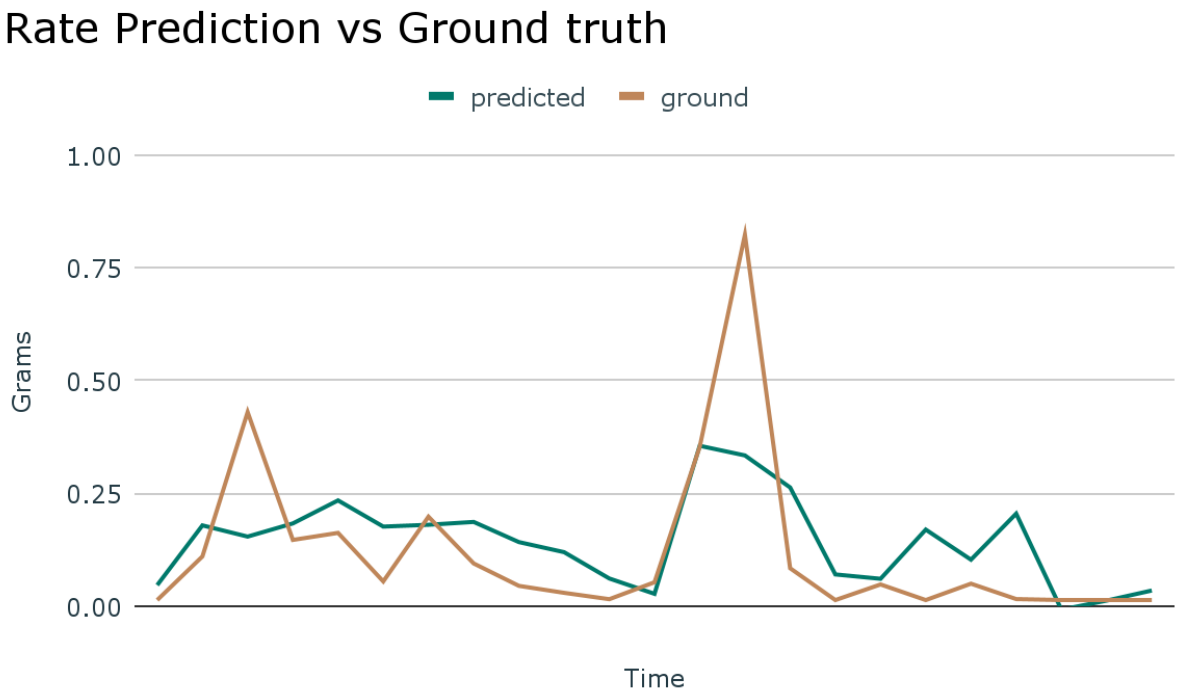
The dataset was divided into three separate groups. First, the data from two of the three rainy days were used to create a training and validation set. The training and validation set came from the same episodes of rain but different spectrograms, 2/3rds for training and 1/3rd for validation. A separate test dataset was created from a third day, i.e episodes of rain not used at all in the training or validation datasets.

The following graph shows the accumulated rainfall expected and predicted for the test dataset.



The average error in accumulation from this data is 8.7e-3 inches. These results are very promising.

Although the accumulated rainfall is the target variable, it is interesting to note that the individual predictions of the rate of rainfall at any moment may be erratic as shown in the graph below.

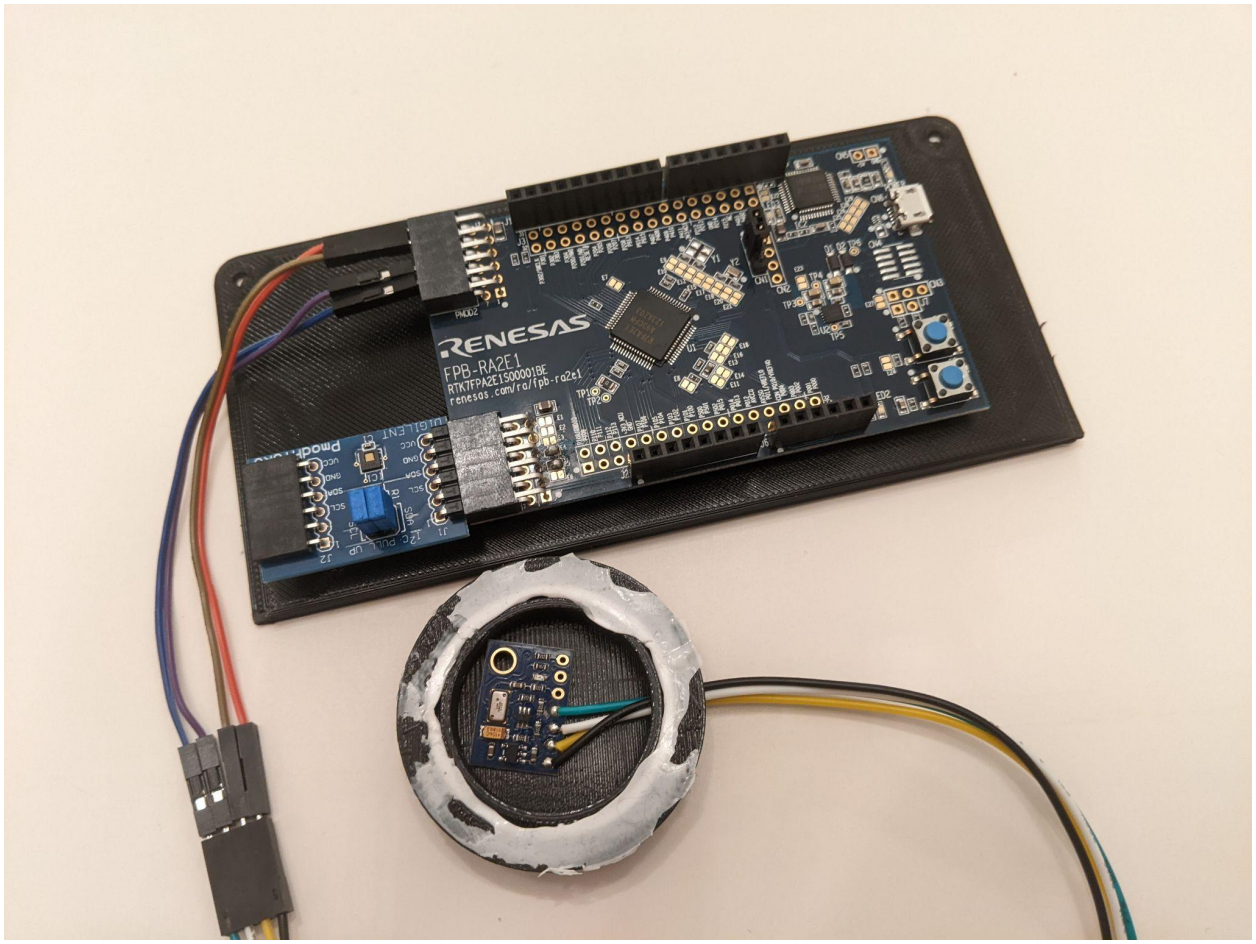


The data in this graph is also from the test dataset. The prediction is frequently well over or under the actual rate, but accumulated over time, a lot of these errors balance out. This is an advantage of summing multiple predictions to calculate the target variable.

The same graphs from the validation dataset show predicted values that follow much more closely with the ground truth. The spectrograms used in the validation set are not used during training, but they are from the same episodes of rain. This suggests that there are some characteristics that vary from one episode to the next and that more data is needed to construct a model that generalizes well.

Meteorological Data

Of course, in addition to rainfall data, we also want to collect other standard measures. For that, we introduce a second board. This board contains the Renesas RA2E1 Risc-V microcontroller. This is a small, very low-power microcontroller that is ideal for interfacing to simple sensors like the ones used here. In particular, the Pmod HYGRO provides both temperature and humidity and the MS5611 is a high-precision pressure sensor.



The board is mounted to the lid (underside) of the project box containing the audio processing board.

The temperature, humidity and pressure are expected measurements from any weather station, but they are also crucial to reducing the overall power usage as described in the following section.

The application code for reading these sensors is available from our github project:
https://github.com/sathibault/smart-weather-station/

Power Optimization

There are two boards used in this project. The microcontroller and environmental sensors (temperature, humidity and pressure) all have fairly low power usage. At full speed, the microcontroller only consumes 5mA and it only takes 1 second to record measurements from the sensors. Running 1 second every minute would consume only 2mAH per day, running for years on a battery.

On the other hand, the audio processing board consumes 17mA and must run for 10 seconds along with the microcontroller to make the rate of rainfall prediction. The results presented in the previous section are based on performing predictions once every 5 minutes. At that rate, the complete system would last less than 6 months on battery. The question is can we use the data collected from the low-power sensors to somehow reduce the number of times the rainfall prediction is made. The answer of course is yes!

There are two parts to the power usage optimization. Part 1 is to reduce the number of times the audio board is used to make a rainfall prediction. This is accomplished by predicting the likelihood of rain and only making the rainfall prediction if the likelihood of rain is over some threshold. For this part, a small artificial neural network is trained with 3 hidden neurons. The inputs are constructed from the aggregated values of temperature, humidity and pressure over a 1 hour window at 5 minute resolution.

Although this rain prediction model saves power by limiting the amount of time the higher power audio processor runs, it also introduces additional error because we may not start measuring rainfall until some time after the rain has actually started. Part 2 of the power optimization is to predict how much rainfall has already accumulated before the part 1 model positively predicted rain. So the procedure is:

1. Collect temperature, humidity and pressure measurements, at least every 5 minutes
2. Use part 1 model to predict if it's currently raining based on environmental measures
3. If the prediction is not raining (score under threshold), go to sleep
4. If the prediction is raining (score over threshold), then run rainfall rate prediction
5. If the rainfall rate prediction is 0, go to sleep
6. If the rainfall rate prediction is > 0, then it is raining, and apply part 2 model to predict the amount of missed rainfall

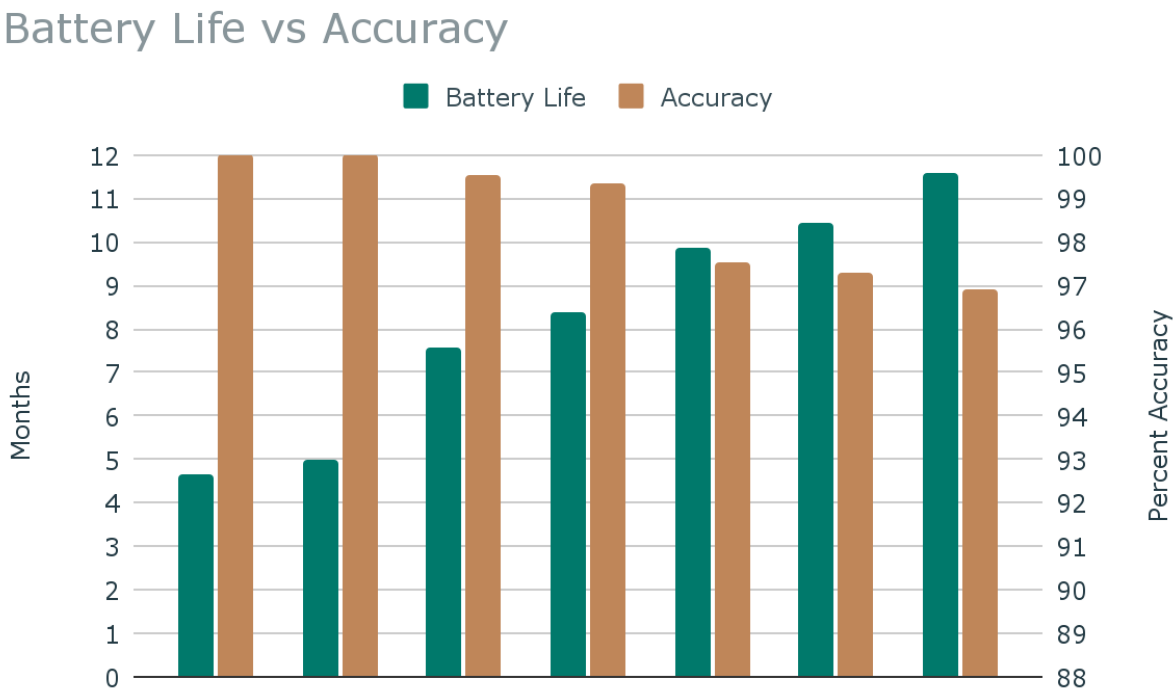
The part 2 model is a simple linear regression model based on the same environmental measures used in part 1. Both of these models are available from our github project:
https://github.com/sathibault/smart-weather-station/tree/main/models

Both models were trained from a public high-resolution smart home dataset available from Kaggle:
https://www.kaggle.com/datasets/taranvee/smart-home-dataset-with-weather-information

Results

The rain prediction and missed rainfall models were used in a simulation to explore the battery life vs accuracy tradeoff. By adjusting the threshold used to make the rain prediction with the part 1 model, we can control the number of false positives (i.e. predicted rain and used more power when it was not raining) vs. how late we are in starting the rainfall measurements (i.e. more error).

The following graph shows the battery life and accuracy for various thresholds. As can be seen, it is possible to extend the battery life to over 8 months with very little impact on overall performance.



Of course, actual battery life will depend on the number of annual rain days which is regional. The test dataset consisted of 1,456 hours of rain across 266 episodes during the year.

False Start

At the beginning of this project, we wanted to measure rainfall directly using an image sensor. Image sensors can consume a lot of power, but the same Vision FPGA SoM used above also supports a low-power image sensor. The idea was to use motion detection to capture rain drops in an image which would appear as streaks in the image. The streak width would be proportional to the speed and size of the drops. This information should be sufficient to train a regression model to predict the rate of rainfall.

We built a prototype and collected some data and the initial results proved that we could capture rain drop streaks in this way. However, after collecting a number of days worth of data, we discovered that it was not working adequately, and we were not able to predict rainfall from the images. In the end, it was determined that the low-cost and low-power sensor simply has too much noise to sufficiently separate the rain motion from the noise.

Although it did not work out with the sensor used, I believe this is still a viable approach, but the cost to obtain necessary sensor quality both in monetary terms and in power usage would be too high. However, as prices come down on event-based sensors, this approach may be feasible in the future.

Conclusion

The Tiny Cow smart weather station prototype demonstrates the feasibility of building a low-cost and low-power weather station capable of measuring rainfall accumulation at sub-hourly resolution and with no moving parts. Through the use of a very tiny convolutional neural network (only 2,800 parameters), rainfall accumulation can be predicted solely from microphone audio data. Although the accumulated rainfall prediction consumes significantly more power than typical environmental sensors, a simple classification model (raining or not raining) was trained on the low-power sensor data to reduce the number of times the higher-power rainfall prediction needs to be performed. Furthermore, a linear regression model is trained to compensate for some of the errors introduced by starting the rainfall prediction too late due to incorrect predictions made by the raining/not raining model. With this power optimization, the system is expected to run up to 8 months on battery power with less than 1% error. The total cost for the major board components (sensors and compute) is under $50.