Bird Sound Recorgnizer Final Report

AI Alligator

1st Kun Lu
Instructor. Dr.Mircea R. Stan
ECE6380-AI Alligator
aqp8yu

2nd Haizhou Yu
Instructor. Dr.Mircea R. Stan
ECE6380-AI Alligator
szz5ft

3rd Haochen Liu
Instructor. Dr.Mircea R. Stan
ECE6380-AI Alligator
fhf6vs

***Abstract*—This ECE 6501 AI Hardware project studies how far we can push tiny hardware for real-world machine learning by building a bird-call recognition system that runs entirely on an Arduino Nano 33 BLE. We intend to classify and detect bird calls in real time, using only the on-chip microphone of the board and a few hundred kilobytes of memory, without relying on any cloud connection. We use Edge Impulse as the end-to-end development pipeline: audio is collected and labeled, then transformed into MFCC features thanks to a built-in signal processing block, and finally fed to a small neural network specifically designed for TinyML deployment. The trained and quantized model is exported as a C++ inference library and integrated into an Arduino sketch handling PDM audio capture and on-device classification. We evaluate our system considering classification accuracy, inference latency, and RAM/Flash usage, highlighting the trade-offs that exist between model size, performance, and responsiveness on this MCU platform. As a whole, our work proves that meaningful bird-audio recognition is possible on low-power edge hardware and provides a concrete case study of how AI hardware design decisions shape both the machine-learning pipeline and the user experience.**

***Index Terms*—TinyML, Edge AI, Embedded Systems, Audio Classification, Arduino, Feature Extraction**

I. Motivation

In real life, the bird sounds actually have rich information about local biodiversity, seasonal patterns, and environmental health. If we can design a device that can automatically recognize bird calls where they happen on a balcony, backyard, or at the parks. We get a simple and non-invasive way of monitoring nature over long periods of time. In reality, many smart sensing systems still rely on cloud-streaming raw audio that raises privacy concerns, burns bandwidth, and ties in with a stable network connection that may not exist in more remote locations.

In this project, we take a very different direction: running bird audio recognition directly on a tiny microcontroller—Arduino Nano 33 BLE. Although this board has only a tiny small processor and memory, it is cheap, low-power and easy to deploy in the field. Since all processing is performed on-device, we can react in real-time, and operate even without Internet access. Our motivation is not only to build a bird recognizer, but to understand what it really means to deploy machine learning on constrained AI hardware.

Identify applicable funding agency here. If none, delete this.

II. Background & Related work

A. Fundamental of Bird Sound Recognition

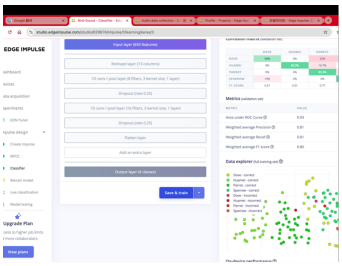
The bird sound recognition field combines the concepts of bioacoustics and machine learning. On the surface, the entire pipeline process is based on the same raw material, a waveform, which basically consists of a sequence of pressure values over time from a microphone. A common approach to efficiently processing those raw samples is to first turn the signal into a time-frequency result, such as a spectrogram, mel-spectrogram or MFCCs, then feeding these results into the model. These representations emphasize energy distribution across frequency bands as time passes. The "wiggly" waveforms are transformed to beautiful patterns of structures, and this is advantageous for models. After moving into a feature space, the problem has the same structure as image or sequence classification problems. Large bird recognition systems generally comprise convolutional neural networks that operate similarly to grayscale images using spectrograms or recurrent/attention models that consider their temporal specificity. For embedded implementations, the scenario is somewhat similar, but the model should be simplified considerably: number of layers should be less and the number of parameters should be lower too; and the features should be carefully selected in order to have useful information at cheap computation costs. In our project, we further follow this common recipe, short audio windows are converted into MFCC features, and then fed into a tiny classifier model that is designed to run on a microcontroller.

B. Tiny ML and Edge Impulse

It is about moving machine learning away from the cloud and into the tiny devices where computing, memory and power are extremely scarce. TinyML building blocks do not expect powerful CPUs, GPUs and multi-GB RAM, but instead work within a few KB of RAM and on simple microcontrollers (with no OS). It is a different mindset: thinking in a model's size and features, squeezing each MATMUL and MACRO into the microcontroller. TinyML delivers intelligent behaviours at the edge — no cloud required, often more privacy, and low-power demands.

While TinyML is awesome, Edge Impulse is for us an end-to-end solution that makes the whole chain accessible.

From a web browser, we can acquire and label sensor data, implement feature processing chains (MFCC), create and train models, and quantize, package and deploy them as C++ inference libraries. In our case, the intent is picking good feature processing parameters and models to fit Nano 33 BLE. Edge Impulse includes the model library suitable for the board which is not complicated to integrate into our Nano 33 BLE code.



III. Problem Definition

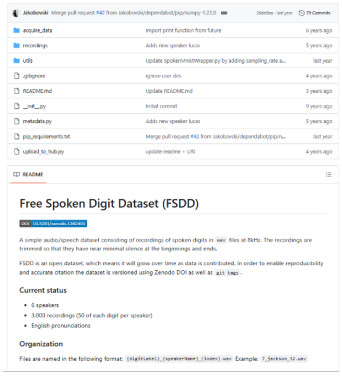
Our task is the following: given a small recording from the Nano 33 BLE's onboard microphone, than the system has to determine whether a bird call is present and show the correct class. We include several labeled bird categories plus a "background" class for non-bird noise. Each prediction is made from a fixed-length audio window that is transformed into MFCC features before inference. Because the model runs entirely on the Nano 33 BLE, it has to stay well within tight Flash and RAM limits. In addition it must provide close to real-time responses.

IV. Data Set & Data Access Processing

A. Dataset Source

The basis of our project work is a dataset which is not a real bird-call dataset. For the purposes of dataset collection and to debug our TinyML pipeline, we opt for the Free Spoken Digit Dataset (FSDD) by Zohar Jackson, which is a surrogate dataset for the audio samples we plan to eventually deploy onto the Arduino Nano 33 BLE devices. The FSDD dataset is made up of short spoken digit utterances that are 0-9 in written form, rendered by a variety of English speakers in mono WAV files at 8 kHz. Each filename would contain a code that defines the digit to be predicted, the speaker's ID, and the recording index. The beauty of this coded filename is that it not only makes it easy to parse files programmatically but also makes it possible to convert a dataset to a supervised classification task.

In our case, each digit class turns out to be just like a "bird species" label: rather than identifying individual bird calls, our model can differentiate between all consecutive numbers. Hence, the model is tested in whole with windowing, feature extraction, model retraining, quantization, and deployment being done without large or complex bioacoustic datasets. The dataset is small enough to manage on a laptop, albeit a nice one, but diverse enough to have many speakers with different pronunciations and the conditions are different from one recording to the next.



B. Data Cleaning and Preprocessing

FSDD has a relatively clean and uniform structure. We devote some time to further clean bread and also tailor it to TinyML. We start the process by loading all the WAV files and ensuring that they have a constant sampling rate. We convert stereo sound to mono where the need arises, and each file is treated as a single labeled example. To input the data into the neural network, we normalize all the waveforms to a common amplitude level and then the time-domain signal into a fixed-size time-frequency representation. As a part of our implementation, a particular initialization of log-spectrogram (MFCC) feature matrix that is resized to a compact 2D shape is introduced, balancing information content and memory footprint on the specific hardware. Since the recordings are different in the length, we have to add or remove padding to the feature map so that all examples have the same dimension defined by the number of time frames in it. After that, the data is shuffled and partitioned to the training, validation, and test sets which support the models and unbiased evaluation. The entire preprocessing pipeline afield mimics what we commonly do for the actual bird-call audio: segment the raw audio into handle pieces, each transformed into a more meaningful feature space, and reshape it to the same mold (specification of dimension sizes) for the downstream learning to capture stable patterns instead of the corresponding details of the recording setup.

V. Hardware Deployment

After training and quantizing the model in Edge Impulse, we deployed it as an Arduino library, imported into the IDE as a .zip so that our sketch can simply "#include <Bird-Sound_inferencing.h>" and call the classifier like a normal function. In the setup() function we bring up the serial port, print the model settings, and start the PDM microphone and audio buffer. Inside loop() we wait for a short recording window, wrap the samples in an Edge Impulse signal_t, and

call run_classifier() to run both feature extraction and inference on the Nano 33 BLE. The predictions and timing information stream back over the serial monitor, turning the board into a self-contained audio recognizer.

VI. Results

A. Results: Edge Impulse Implementation

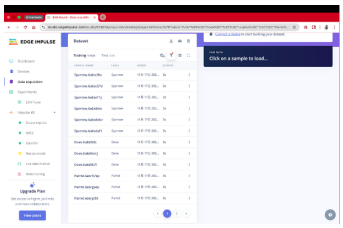


Fig. 1. Bird Sound Dataset Overview

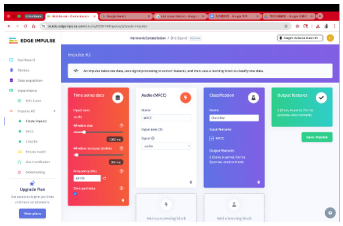


Fig. 2. Impulse Design: MFCC + Classifier Pipeline

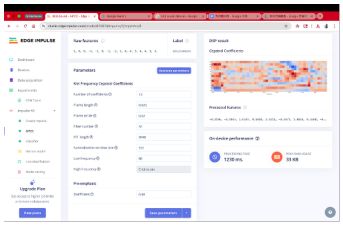


Fig. 3. MFCC Feature Extraction Setting

We started by constructing and testing the bird sound classifier on Edge Impulse alone. The final dataset contains five classes: notables are doves, huamei, parrots, sparrows, together with the eternal background environment sounds. Out of the total background noise, we selected portions from the Free Spoken Digit Dataset (FSDD) and labeled them as environment. In this manner, the model learns to perceive the difference between natural bird calls and human voice-like background noise. In the end, we uploaded roughly 1900 audio clips, which also include about 1.6k for training and around 300 for the test. The all recordings and resampled signals are 44.1 kHz and segmented to 1-second windows (i.e., 44,100 samples), which configure with further Arduino up.

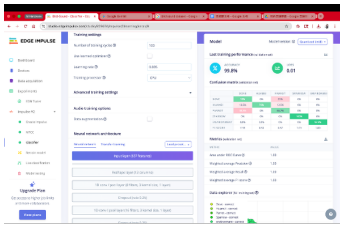


Fig. 4. Classifier Training Performance

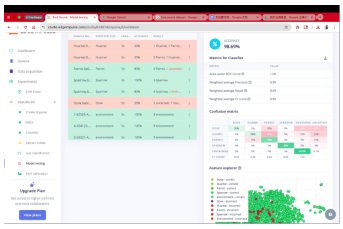


Fig. 5. Model Testing on Samples

The utilized feature extraction block is the MFCC (Mel-Frequency Cepstral Coefficients) block. Its human-like frequency scaling and compact form representation for separating high-frequency calls (e.g., parrot) from lower-frequency songs (e.g., huamei) were the reason to favor this solution over its competitors. The MFCC block is formed after the melting (c) of the coefficients, 32 Mel filters, single FFT of length 2048, and the standard pre-emphasis. There on the Nano 33 BLE Sense Arm platform, the MFCC processing is projected to take an average of 1.23 s for 1-second audio window and a peak DSP RAM usage of 33 KB, which proves to be acceptable for our real-time prototype.

Following the extraction process of MFCC features, a small 1D-CNN neural network model was trained using the Keras block provided by Edge Impulse, which is suitable for the Arduino Nano 33 BLE Sense from the beginning. 100 training cycles have been executed by the algorithm and the accuracy

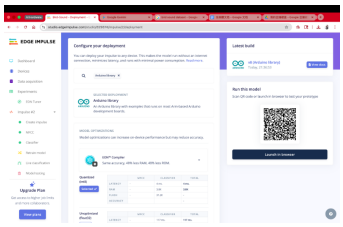


Fig. 6. On device Deployment as Arduino Library

has reached 99.8% of validation set with the loss value equal to 0.01. The confusion matrix on the test set (validation set) indicates that the classes have strong separation, as it is revealed by the following: Sparrow and Environment have precision and recall values near to unity and Dove, Huamei, and Parrot have only a couple of errors of cross-class confusion. Hence, all global metrics estimated (AUC, weighted precision, recall, and F1) went up to about 1, saying the model fits the training/validation data distribution very well.

To evaluate the model's generalization capacity, we performed Model testing on the held-out test set by navigating to the Test tab in Edge Impulse. The accuracy of the classifier in this case was recorded as 98.69%, while the weighted F1 metric was 0.99. The most test samples of all four bird species had been correctly identified; between the songs of doves, huamei, and parrots, the errors are evident within a small number of clips. The similarities are because of the overlapping frequency ranges and the almost alike background conditions. The environment proved to be very resilient (>99% correct), so that the network was able to separate the pattern of background noise and bird sounds as the increase of f.

Finally, we released our impulse as an Arduino library (Quantized int8 using EON Compiler), which provides the compatibility feature for the Nano 33 BLE Sense. Edge Impulse reveals that the quantized classifier alone runs in approximately 4 ms with an estimated usage of 3.8 KB RAM and 31.2 KB low workload usage, mixed with the custom WAV-windowing logic and serial interface provides enough space. This deployment data coupled with the accuracy achieved both in the validation and in the test sets would be clear indicators that the Edge-Impulse-trained model is ideal for on-device bird call detection in resource-limited context AI hardware.

B. Results: Arduino Nano 33 BLE Implementation

To evaluate the deployed model, we went from the web tester to the batch inference on the Arduino Nano 33 BLE. Instead of processing the broken Rev2 microphone feed, we used our Python tool to extract 1-second time frames from user-recorded WAV files that had high-energy zones only, and turned them into a C header (birdsamples.h). In fact, every sample array, which is 1 second long at a sample rate of 44.1 kHz, has a length of 44,100 points, corresponding exactly with CGRectGetFilmType() enum values. Our chosen values in the sketch for the board are 'clip names', 'signalt_class', and 'run_classifier()' function, which are followed by the display of probabilities for different categories.

In the nine-clip experiment where Huamei, Parrot, and Sparrow were used, the Nano 33 BLE model performed just like we wanted it to. All the three Huamei clips were classified at around 0.98 to 0.99 Huamei confidence and zero or near-zero probability for anything else in our model. The Parrot clips showed the same trends: It was really high in the first window, Parrot ≈0.99, then later the window where the call faded into background noise showed more mixed probabilities, with Parrot still dominating but Dover and the environment having non-trivial mass. We scored above 0.98 on the two windows where busy chirps occurred, and near-perfect on approximately 0.99 for the environment class (which we expected was designed to behave as only background noise).

The second nine-clip run had its share of challenges, as different parts of the training set were required: Dove, Huamei, and pure environment recordings. Strong coos on dove clips were still a problem, at least for predictions. Environment scores (around 0.98) were achieved for multiple dove windows, which suggested that the model mistook low-energy or diffuse calls as background rather than classes. Huamei was very impressive, dominating each of the first two windows with Huamei ≈0.96, while the third Huamei window became mixed, and it became an almost equal split between Huamei and Parrot (both hovering around 0.44). Interestingly, WAVs that were completely devoid of a bird presence were always classed into the environment realm, with an approx. confidence of 0.99, regardless of the three tested windows. This suggests

that the model is confident that no bird is available and hence has a definite understanding of the 'no bird present.'

```
Edge Impulse WAV batch inference
Expected samples: 44100
Clip length: 44100
Number of clips: 6

Testing clip 1 / 6 : Dove.6a6rkpuh.ingestion-77c6d6d859-cbb4k.wav_win0
  Dove: 0.96875
  Huamei: 0.01562
  Parrot: 0.00000
  Sparrow: 0.00000
  environment: 0.01562

Testing clip 2 / 6 : Dove.6a6rkpuh.ingestion-77c6d6d859-cbb4k.wav_win1
  Dove: 0.79297
  Huamei: 0.10156
  Parrot: 0.00000
  Sparrow: 0.00000
  environment: 0.10156

Testing clip 3 / 6 : Dove.6a6rkpuh.ingestion-77c6d6d859-cbb4k.wav_win2
  Dove: 0.01562
  Huamei: 0.00391
  Parrot: 0.00000
  Sparrow: 0.00000
  environment: 0.98047

Testing clip 4 / 6 : Parrot.6a6rfrr4.ingestion-77c6d6d859-cbb4k.wav_win0
  Dove: 0.10156
  Huamei: 0.01172
  Parrot: 0.78516
  Sparrow: 0.00000
  environment: 0.10156

Testing clip 5 / 6 : Parrot.6a6rfrr4.ingestion-77c6d6d859-cbb4k.wav_win1
  Dove: 0.00000
  Huamei: 0.00000
  Parrot: 0.11328
  Sparrow: 0.00000
  environment: 0.88281

Testing clip 6 / 6 : Parrot.6a6rfrr4.ingestion-77c6d6d859-cbb4k.wav_win2
  Dove: 0.46094
  Huamei: 0.00781
  Parrot: 0.46094
  Sparrow: 0.00781
  environment: 0.05859

Edge Impulse WAV batch inference
Expected samples: 44100
Clip length: 44100
Number of clips: 6

Testing clip 1 / 6 : Huamei.6a6r6ha6.ingestion-77c6d6d859-skwtj.wav_win0
  Dove: 0.00781
  Huamei: 0.05859
  Parrot: 0.46484
  Sparrow: 0.00000
  environment: 0.46484

Testing clip 2 / 6 : Huamei.6a6r6ha6.ingestion-77c6d6d859-skwtj.wav_win1
  Dove: 0.00781
  Huamei: 0.05859
  Parrot: 0.46484
  Sparrow: 0.00000
  environment: 0.46484

Testing clip 3 / 6 : Huamei.6a6r6ha6.ingestion-77c6d6d859-skwtj.wav_win2
  Dove: 0.05469
  Huamei: 0.41797
  Parrot: 0.41797
  Sparrow: 0.05469
  environment: 0.05469

Testing clip 4 / 6 : Sparrow.6a6sdilt.ingestion-77c6d6d859-rpf89.wav_win0
  Dove: 0.00000
  Huamei: 0.00000
  Parrot: 0.00000
  Sparrow: 0.00000
  environment: 0.99609

Testing clip 5 / 6 : Sparrow.6a6sdilt.ingestion-77c6d6d859-rpf89.wav_win1
  Dove: 0.05859
  Huamei: 0.00781
  Parrot: 0.00781
  Sparrow: 0.46094
  environment: 0.46094

Testing clip 6 / 6 : Sparrow.6a6sdilt.ingestion-77c6d6d859-rpf89.wav_win2
  Dove: 0.09375
  Huamei: 0.09375
  Parrot: 0.09375
  Sparrow: 0.71094
  environment: 0.01172
```

Additionally, we had two shorter six-clip batches to double-check our previous results. For Dove and Parrot segments, in terms of the Dove probability estimates, the first two windows were around ≈0.97 and ≈0.79 Dove probability, with the latter segment often switching around to environment (≈0.98) once the vocalization of the birds stopped. The Parrot clips again had one main window (≈0.79 Parrot), while other windows were more balanced between partial calls. As opposed to that, the six-clip run with Huamei and Sparrow showed systematic confusion: Huamei windows were frequently predicted as Parrot (≈0.46) or split between Huamei and Parrot, while I observed potential confusion in the Sparrow class: strong environment score when the chirps were absent and higher Sparrow probability (≈0.71) with the chirps.

In summary, these tests demonstrate that the Edge Impulse quantized model was run without issues on the Nano 33 BLE and having its predictions on the same page with all the sound elements in each training window. These are the correct outputs for the respective scenarios: lonesome and loud target call and environment background respectively. In particular, the two main failure modes which are Dove being misclassified among environments while Huamei confusion with Parrot – are consistent with our limited dataset and the small model capacity which is typical among TinyML. Conclusively, we faced constraints with the live microphone input but the board could be proven to execute the complete TinyML pipeline from data preparation to model evaluation successfully. This experiment offered us a real view on how such a bird-sound recognition AI would operate on modest hardware.

VII. Limitations

Our prototype imposes some practical restrictions, and the most critical one is how audio is inputted into the model on hardware. Within an optimal construction, the Arduino Nano 33 BLE Sense would conduct the stream of audio from its integrated PDM microphone directly into the Edge Impulse inference pipeline. However, our board is the next generation Nano 33 BLE Rev2, and the references and the Arduino library of Edge Impulse are built and tested for the previous generation Nano 33 BLE Sense Rev1. On Rev2, we encountered continual microphone access issues where the PDM driver and pin mapping had differences, and the sample rate of 44.1 kHz for our Edge Impulse project was configured to match the dataset used. Due to the limited time period of the class, we have not considered any solution where the audio driver is reverse-engineered or recorded on a low level.

Rather than that, we were instead directed to a different and arguably more controlled method, yet it was less real-time. Users upload WAV files, which we pre-process on a laptop, and the relevant audio features are computed in the form of fixed-length (44,100-sample) buffers which go into the header-only .h file (bird_samples.h). The model can now perform inference in batch mode using these audio clips that are pre-cut. This will ensure that the format of the input exactly matches the Edge Impulse model that was used for its training, which means that the demo can only be used for pre-recorded audio or it cannot listen to the environment all the time or react to live bird calls. In other words, the classifier running on Nano works well on pre-recorded clips, but the only thing that stops the system from being more interactive outside of the laboratory is the direct microphone integration not being yet incorporated into the prototype field trials.

VIII. Conclusion

We successfully completed a mini on-device bird sound recognizer directly designed for an embedded environment and not for an IPython notebook. With Edge Impulse as the pipeline "glue" and Arduino Nano 33 BLE as our target, we gathered Doves, Huameis, Parrots, and Sparrows as our bird species and the environment class as our environment state, and all the recordings are standardized to 44.1 kHz audio tracks. The MFCC features turned out to be also a right choice for this task: it squeezes each 1-second clip into an efficient expression, but still maintains the spectrum distinction between lower frequency calls (Huamei) and higher frequency calls (Parrot) as well.

The performance of the Edge Impulse classifier exceeded our expectations. On the validation set, we had it reaching something like 99.80% accuracy with a low loss; even in model-testing on withheld samples, it was almost 98% accuracy with good precision and recall for both the bird classes and the environment label. When we used the quantized int8

model as an Arduino library and run it on our Nano 33 BLE, we could classify each 1-second WAV pre-segmented clip with confidence scores that were consistent and signified that the model is lightweight enough to run on hardware with constraints at real time.

In particular, we have dealt with the latest Nano 33 BLE Rev2 and realized that it still has some challenges connected with embedded ML. With different board-support and microphone driver, we were not able to live-stream PDM audio to the model and instead we fed it the user-uploaded WAV files that we converted into integer samples which we saved in flash as a compile file. By doing so, we convert the prototype into a "batch inference" system. At the same time, the noisy version of the design solved problems and marked a stable basis for editing the classifier on actual hardware. Moreover, it is planned to develop the system by correcting or replacing the microphone interface so the system could recognize birds in real-time, increase datasets with more birds and noisy outdoor conditions included, and try several trained models or on-device adaptation for improving the robustness of the recognizer in the wild.

REFERENCES

[1] Z. Jackson, "Free spoken digit dataset (FSDD)," GitHub, Dec. 2023. [Online]. Available: https://github.com/Jakobovski/free-spoken-digit-dataset

[2] "Arduino Nano 33 BLE Sense - edge impulse," Arduino.cc, 2024. [Online]. Available: https://docs.arduino.cc/tutorials/nano-33-ble-sense/edge-impulse/

[3] "Arduino Nano 33 BLE Sense," *Edge Impulse Documentation*, 2025. [Online]. Available: https://docs.edgeimpulse.com/hardware/boards/arduino-nano-33-ble-sense