WRENNET FOR BIODCASE-TINY 2026 TASK 3

Technical Report

Leonardo Mannini, Stefano Ciapponi, Elisabetta Farella

Fondazione Bruno Kessler
Trento, Italy

ABSTRACT

This report describes our BioDCASE-Tiny 2026 Task 3 submission based on the WrenNet architecture for low-power bird sound classification. The model was trained for the challenge through Xeno-canto pretraining on the target species followed by fine-tuning on the official BioDCASE development data. For embedded deployment, the waveform model was adapted to the official submission framework and split into a frozen frontend/backbone path and a stateful recurrent step suitable for TensorFlow Lite Micro on ESP32-S3 hardware. On the 549-file development validation split, the PyTorch reference, host TensorFlow Lite model, and embedded split TensorFlow Lite path produced identical top-1 predictions, with 65.21% accuracy and 0.933 ROC-AUC. The embedded benchmark fits within the available memory and flash budget, with a peak TensorFlow Lite Micro arena usage of 263.5 kB and approximately 4.46 s total processing time per 3 s clip on the tested ESP32-S3 board. The deployable embedded path uses a 205 kB float32 backbone model and a 118 kB float32 recurrent-step model.

***Index Terms*—** Bioacoustics, bird sound classification, TinyML, ESP32-S3, TensorFlow Lite Micro

1. PROJECT OVERVIEW

The BioDCASE Tiny Challenge 2026 (Task 3) addresses the critical real-world need for low-power, non-invasive biodiversity monitoring by deploying automated birdsong classification directly onto edge devices. The core constraint of this challenge is balancing high classification accuracy across 10 distinct bird species, plus an urban background noise class, with the memory and processing limitations of an ESP32-S3-Korvo-2 microcontroller-class target [2, 3].

As the authors of WrenNet, our objective was to validate the model's behavior and efficiency on a standardized bioacoustics dataset. Each raw 24 kHz, 16-bit PCM mono recording is converted to a fixed 3 s waveform, peak-normalized, and processed by a custom spectral frontend. The frontend performs an STFT-based filterbank projection, compresses the signal to 64 bins, applies sample-level normalization, and feeds the classifier with a 301×64 sequence. Because WrenNet inherently prioritizes a low hardware footprint, and because we designed the model to be explicitly stateful and recurrent, deploying it onto ESP32-S3 target hardware required a custom submission integration and an embedded inference split. This architectural choice bypassed the default baseline feature extraction and model path, allowing us to analyze how WrenNet balances accuracy, memory footprint, and latency under the BioDCASE-Tiny evaluation constraints.

For hardware validation we used an ESP32-S3 development board available in our lab instead of the full ESP32-S3-Korvo-2 audio board. This board does not test the Korvo-2 microphone/audio path, but it uses the same ESP32-S3 family and comparable flash/PSRAM constraints, making it a useful proxy for TensorFlow Lite Micro allocation, kernel compatibility, build/flash behavior, and parser-compatible timing output.

2. SOLUTION DESIGN AND MODELING

2.1. Model

To meet the stringent hardware constraints of the ESP32-S3 microcontroller, this solution leverages WrenNet, a hybrid neural network architecture engineered for continuous, multi-species bioacoustic streaming on low-power edge devices [1]. We adopted only the WrenNet architecture and trained it for this challenge using Xeno-canto pretraining on the BioDCASE target species followed by fine-tuning on the official BioDCASE development data.

The model is organized as three stages. The first stage replaces static mel-scale filterbanks with a differentiable frequency-warping frontend. This frontend uses a parametrized transition between logarithmic and linear frequency spacing, allowing spectral resolution to be adapted to avian vocalizations. In the submitted system the learned frontend is frozen for inference. The resulting feature sequence is processed by a lightweight Matchbox/PhiNet-style encoder based on causal depthwise separable 1D convolutions, squeeze-and-excitation modules, dropout, and skip connections. Temporal modeling is performed by a single unidirectional GRU, which carries temporal context through its hidden state. A linear projection and learned temporal attention layer then aggregate the sequence into a fixed-size context vector for the final 11-class classifier. In compact form, the architecture is: adaptive spectral frontend, causal convolutional encoder, streaming GRU state update, attention pooling, and linear classifier.

The final PyTorch checkpoint contains 57,552 trainable parameters. The model input is a 3 s waveform and the output is a vector of 11 logits corresponding to the 10 target bird species plus the background class.

2.2. Hyperparameters and Tuning

The final configuration was selected from validation performance and export feasibility. The model uses the adaptive log-linear spectral frontend described above. For the submitted model, waveform clips are 3 s long at 24 kHz. The frontend uses 64 filters, FFT size 1024, hop length 240 samples, and a 150 Hz–12 kHz frequency range, producing a 301×64 feature sequence. The frontend breakpoint and transition-width parameters were initialized at 4000 Hz

and 100 respectively in the BioDCASE configuration. The convolutional encoder uses 32 base filters and a 3-layer Matchbox-style stack, and maps the 64-bin frontend sequence into a 32-channel frame embedding. The recurrent stage uses a single unidirectional GRU with hidden size 70. During tuning we prioritized configurations that preserved the original WrenNet streaming structure while remaining exportable to TensorFlow Lite Micro without unsupported recurrent operators or excessive tensor allocation.

2.3. Additional Data

The official BioDCASE development data were used with the provided split: 2,200 training clips, with 200 clips per class, and 549 validation clips, with 50 clips for each class except 49 for Great Tit. All clips are 3 s, 24 kHz mono recordings, corresponding to 1.83 h of official training audio and 0.46 h of official validation audio. The background class was taken only from the official BioDCASE data and was not synthesized from Xeno-canto.

We used Xeno-canto bird recordings [4] as external pretraining data. The processed Xeno-canto subset contains 55,227 non-background clips from the ten target species, corresponding to 46.02 h of 3 s audio. It was split into 49,705 training clips and 5,522 validation clips. To keep the pretraining label space identical to the BioDCASE task, 250 official background clips were added to this pretraining split, giving 55,477 total clips or 46.23 h. The per-class Xeno-canto clip counts were 7,596 Common Chaffinch, 7,511 Common Chiffchaff, 7,829 Eurasian Blackbird, 5,486 Eurasian Blackcap, 4,844 Eurasian Blue Tit, 2,889 Great Spotted Woodpecker, 8,502 Great Tit, 1,294 Mallard, 5,765 Song Thrush, and 3,511 Tawny Owl.

The Xeno-canto recordings were filtered by target species and challenge cutoff date, decoded, converted to mono, resampled to 24 kHz, and segmented into 3 s clips. The snippet extraction pipeline applies a 150 Hz–10 kHz band-pass filter for energy tracking, centers clips on detected high-energy vocalization peaks, and falls back to the maximum-energy 3 s window when no stable peak is found. The resulting pretraining files are 16-bit mono WAV clips with 72,000 samples. The Xeno-canto material consists of natural field recordings, so background species, habitat, and SNR variability are retained from the source recordings; no explicit habitat or SNR stratification was used.

2.4. Training

Training and fine-tuning were performed in PyTorch on 3 s, 24 kHz mono waveform clips using the official 11-class label set. The submitted model follows a two-stage transfer-learning setup. First, a WrenNet model initialized for this task was trained on the external Xeno-canto clips for the BioDCASE target species to learn a challenge-specific bird-acoustic initialization. Second, the resulting checkpoint was used to initialize the BioDCASE model, which was fine-tuned on the official development data with the challenge label set, waveform duration, and background class. Thus the original WrenNet work provides the model design, while the submitted weights come from our Xeno-canto pretraining and BioDCASE fine-tuning pipeline.

Fine-tuning used focal loss with $\gamma = 2.0$ and uniform class weights, AdamW with learning rate $8 \cdot 10^{-4}$ and weight decay $1.5 \cdot 10^{-2}$, and a plateau scheduler monitoring validation loss. The maximum training length was 75 epochs, and the submitted checkpoint was selected by validation macro F1; the exported checkpoint corresponds to epoch 19 of the BioDCASE fine-tuning run. Data augmentation was applied only to training waveforms. It consisted of random amplitude scaling sampled between 0.85 and 1.15, additive Gaussian noise with standard deviation 0.005, and random temporal shifts up to 0.05 s. Validation and export runs used deterministic waveform loading without augmentation. After model selection, the same frozen frontend and classifier weights were exported to the host and embedded inference representations so that PyTorch, host TensorFlow Lite, and embedded TensorFlow Lite paths could be compared against the same trained system.

2.5. Inference Configuration and Framework Integration

The official BioDCASE-Tiny framework expects each solution to provide an inference handler, model artifacts, metadata, and optionally generated embedded code. Since WrenNet uses a custom waveform frontend instead of the baseline feature extraction pipeline, we implemented a custom integration layer while preserving the official evaluation interface.

The training code is implemented in PyTorch because it gives direct control over the WrenNet frontend, recurrent classifier, focal-loss training loop, and checkpoint selection. After training, the selected PyTorch checkpoint is treated as the reference model. For submission and deployment we then export the same waveform-to-logits computation through ONNX and convert it to TensorFlow Lite. This produces the host waveform TFLite model: a monolithic float32 graph that receives a 3 s waveform tensor of shape $[1, 72000, 1]$, outputs an 11-class logits vector of shape $[1, 11]$, and has a file size of 4.76 MB. This model is used to check that the exported graph preserves the PyTorch predictions inside the official host-side test pipeline.

For embedded testing, the submission includes an ESP-IDF firmware project based on the challenge template. The monolithic waveform TFLite export is useful for host validation, but it is not the most robust representation for TensorFlow Lite Micro on ESP32-S3 because the recurrent part becomes a large expanded graph. In practice, this unrolled recurrent representation stresses TensorFlow Lite Micro allocation and kernel support, especially through large expanded recurrent subgraphs and concatenation-heavy intermediate tensors. We therefore split the embedded inference path after the frozen frontend. The first TFLite artifact is a backbone model that maps the normalized 301×64 frontend features to a 301×32 sequence of frame embeddings. The second artifact is a stateful recurrent step that is invoked once per frame while the firmware carries the GRU hidden state and temporal-attention accumulators explicitly. This is a deployment refactoring rather than a different classifier: the frame sequence, GRU equations, attention accumulation, and final logits match the PyTorch reference computation. The split keeps the embedded path functionally equivalent to the host model while avoiding the allocation and kernel issues observed with the fully expanded recurrent graph.

The submitted firmware uses two float32 TFLite artifacts: a 205 kB backbone model and a 118 kB streaming-step model. An int8 classifier-only export was tested, but its recurrent layer required a fully unrolled graph. On ESP32-S3 this representation hit TensorFlow Lite Micro and ESP-NN allocation/kernel limits, mainly around expanded recurrent tensors and concatenation operations. For this reason, the final embedded path keeps the stateful split in float32.

The embedded integration keeps the challenge logging format for setup time, feature extraction time, model execution time, out-

put CRCs, and TensorFlow Lite Micro memory allocation. The firmware builds and flashes as a standard ESP-IDF application and prints the measurements expected by the challenge parser.

3. RESULTS

3.1. Final Performance and Metrics

We evaluated three representations of the same trained system on the 549-file BioDCASE development validation split. The PyTorch reference is the original training checkpoint and is used as the numerical baseline. The host waveform TFLite model is the ONNX-to-TFLite export of the complete waveform-to-logits graph and verifies the official host submission path. The embedded split TFLite path is the ESP32-S3/TFLM version, where the same classifier is executed as a backbone plus stateful recurrent step. Performance was measured on this split using top-1 argmax predictions and one-vs-rest ROC-AUC from the 11 logits. All three representations obtained 0.6521 accuracy; host waveform TFLite and embedded split TFLite also had 1.0000 top-1 agreement with PyTorch, and PyTorch and host TFLite both obtained 0.933 ROC-AUC. File sizes are 346.7 kB for the PyTorch checkpoint, 4.76 MB for the host waveform TFLite graph, and 205 kB plus 118 kB for the embedded backbone and recurrent step.

The confusion matrix below shows the validation class decisions for the same run. It is computed from the PyTorch reference predictions; since host waveform TFLite and embedded split TFLite have 1.0000 top-1 agreement with that reference, the same class-level decisions apply to all three inference paths. The strongest classes are Background, Common Chiffchaff, Great Spotted Woodpecker, Mallard, and Tawny Owl.

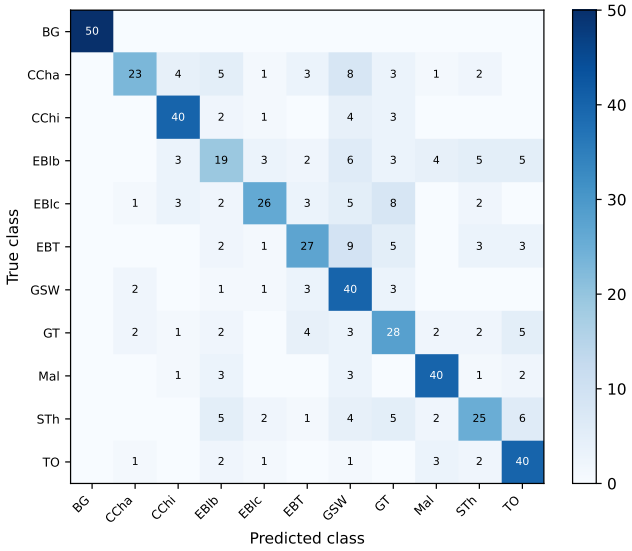

Figure 1: Confusion matrix on the 549-file validation split. BG=Background, CCha=Common Chaffinch, CChi=Common Chiffchaff, EBlb=Eurasian Blackbird, EBlc=Eurasian Blackcap, EBT=Eurasian Blue Tit, GSW=Great Spotted Woodpecker, GT=Great Tit, Mal=Mallard, STh=Song Thrush, TO=Tawny Owl.

The final ESP-IDF application was built, flashed, and monitored on an ESP32-S3 board with 8 MB flash and 8 MB PSRAM. Embedded time and memory were measured from the firmware logs. The timing blocks are placed around setup/allocation, frontend preprocessing, and model execution; model execution is the sum of the backbone invocation and the streaming recurrent-step loop. Memory is reported from the TensorFlow Lite Micro RecordingMicroAllocator, using the largest observed arena allocation.

Table 2: Embedded runtime and memory comparison with the challenge baseline example log.

On-device measurement	WrenNet	Challenge baseline log
Setup and allocation	21.53 ms	2.18 ms
Frontend preprocessing	477.45 ms	3.44 ms
Model inference	3961.88 ms	1324.72 ms
Total per 3 s clip	4460.87 ms	1330.34 ms
Peak TFLM arena allocation	263.5 kB	288.9 kB

On the tested ESP32-S3 configuration, processing one 3 s clip takes longer than real time. The baseline values are taken from the example embedded log distributed with the challenge repository and are included as an indicative runtime reference rather than a like-for-like validation comparison. Our latency is mainly due to the final float32 stateful split: the backbone and recurrent step are executed by two TensorFlow Lite Micro interpreters, and the recurrent step is invoked once per frontend frame. We selected this representation because the int8 unrolled export did not provide a reliable embedded execution path with the available TensorFlow Lite Micro and ESP-NN kernels.

4. CONCLUSION

This submission validates WrenNet on the BioDCASE-Tiny 2026 Task 3 benchmark and provides a complete integration with the official submission interface. The system includes a custom waveform inference pipeline, a monolithic host-side TFLite reference model, and a deployable ESP-IDF embedded path based on a stateful split of the classifier. The final embedded path obtains 65.21% accuracy and 0.933 ROC-AUC on the official development validation split.

The embedded results show that the model fits within the target memory and flash constraints; the main remaining optimization target is latency, especially in the recurrent step and TensorFlow Lite Micro operator path.

5. REFERENCES

[1] S. Ciapponi, L. Mannini, J. Scanferla, M. Anderle, and E. Farella, "Enabling Multi-Species Bird Classification on Low-Power Bioacoustic Loggers," in *ICASSP 2026 - 2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2026, pp. 15212–15216, doi: 10.1109/ICASSP55912.2026.11463679.

[2] BioDCASE, "BioDCASE 2026 Task 3: Bioacoustics for Tiny Hardware," `https://biodcase.github.io/challenge2026/task3`.

[3] C. Walter, Y. Benhamadi, T. Seidel, G. Carmantini, and S. Kahl, "BioDCASE-Tiny 2026: A Framework for Bird Species Recognition on Resource-Constrained Hardware," GitHub repository, 2026. `https://github.com/birdnet-team/BioDCASE-Tiny-2026`.

[4] Xeno-canto Foundation, "Xeno-canto: Sharing bird sounds from around the world," `https://xeno-canto.org/`.