BirdCLEF 2025 Final Report

Reese Mullen Vajinder Kaur Taha Ababou Paul Moon

Executive Summary

This report documents our local inference pipeline for the BirdCLEF 2025 challenge, focused on detecting bird species from soundscape audio. It is based on a high-performing Kaggle notebook with several enhancements.

Key highlights: - **Model**: EfficientNet backbone with FocalLossBCE, trained across 4 folds. - **Input**: 5-second mel spectrogram segments created from 32 kHz soundscapes. - **Prediction**: Ensemble inference with optional test-time augmentation. - **Output**: Ranked species predictions saved to `submission.csv`. - **OOF Scores**: Averaged around **0.962**, indicating strong validation performance.

BirdCLEF 2025 Inference Report

This report documents the inference phase of our solution for the BirdCLEF 2025 challenge. The goal of this competition is to automatically detect bird species from audio soundscapes using machine learning techniques.

We base our work on a high-performing public notebook and introduce modifications to improve prediction accuracy. This file focuses specifically on how we structured the inference pipeline, handled audio preprocessing, implemented the prediction model, and evaluated results.

Key components of this report include: - Overview of the data and preprocessing steps - Description of the model inference pipeline - Visualizations to understand prediction distributions - Summary of detected species and confidence thresholds

**Note : We are using a public notebook as a baseline with some edits of our own to improve the accuracy. This file contains additional edits.

INFO

This report presents the results of a local inference pipeline for the **BirdCLEF 2025 challenge**, which involves identifying bird species from soundscape audio using deep learning.

We adapted and extended a high-performing public Kaggle notebook originally developed by Kadircan İdrisoğlu, incorporating: - A local training and inference setup using EfficientNet models - Custom audio preprocessing (mel spectrograms, test-time augmentation) - An ensemble of 4 models trained using FocalLossBCE - A post-processing step to smooth predictions across time segments

Key Metrics:

- Out-of-fold validation scores:
  - Fold 0: 0.9652
  - Fold 1: 0.9605
  - Fold 2: 0.9607
  - Fold 3: 0.9626
  - **OOF Average:** 0.9622

In [1]:

In [2]: ##FocalLossBCE Use To address the inherent class imbalance in multi-label bird call detection, we implemented a custom loss function named FocalLossBCE. This function combines two components:

Binary Cross-Entropy (BCE): A standard loss for binary classification that penalizes incorrect predictions.

Focal Loss: A more advanced loss that down-weights easy examples and focuses training on harder, misclassified ones.

Our implementation allows tuning of:

alpha: Balances the importance of positive vs. negative examples.

gamma: Controls how much to focus on difficult examples.

bce_weight and focal_weight: Let us blend standard BCE and Focal Loss to benefit from both.

This combined loss is particularly well-suited for imbalanced multi-label problems like ours, where many species appear infrequently but should still be detected accurately. The use of torchvision.ops.focal_loss.sigmoid_focal_loss ensures numerical stability and efficient computation during training.

(See Block 4 in appendix for code)

In [3]:

Visual Index

Configuration Setup

The `CFG` class defines all key hyperparameters and file paths used during the inference stage. This includes audio processing parameters such as FFT size, mel bins, and sampling rate, along with model configuration and file paths for the taxonomy and soundscape data.

Important settings include: - **Sampling rate**: 32,000 Hz - **Mel Spectrogram size**: 512 bands - **Inference threshold**: 0.5 for classifying bird presence - **Window size**: 5 seconds for segmenting soundscapes

(See Block 5 in appendix for code)

In [4]:

Data Summary: Taxonomy

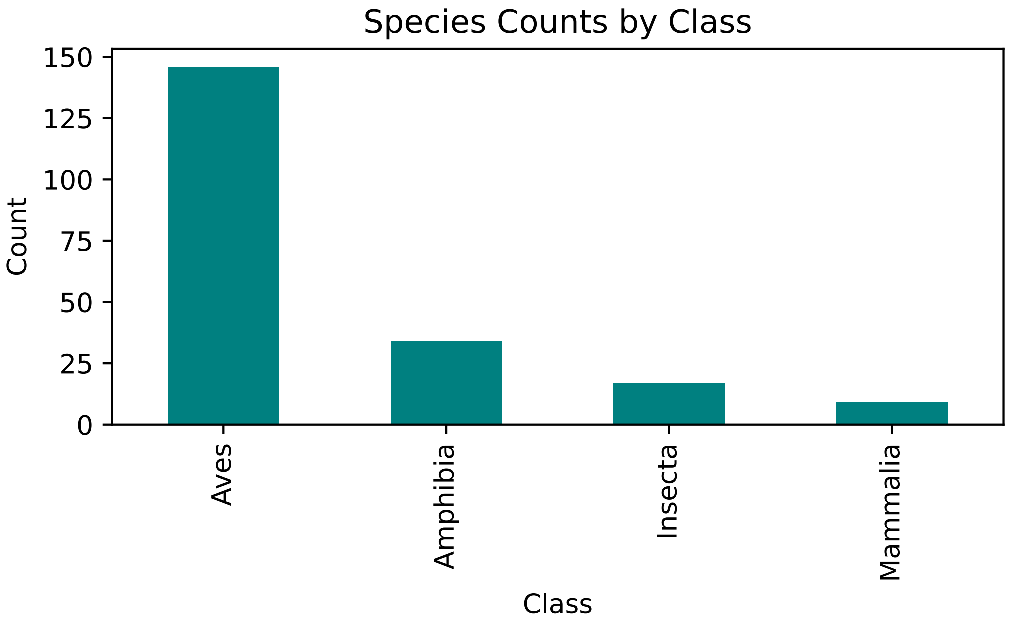
The taxonomy dataset provides the full list of bird species considered in the BirdCLEF 2025 challenge, along with their scientific classifications. Each row represents a species, identified by a unique primary_label, and includes metadata such as common names, scientific names, family, order, and class. This information is critical not only for interpreting prediction outputs but also for understanding how species are distributed across biological groupings. We summarize key properties of the dataset—like total number of species and the number of distinct classes—and visualize class-level distributions to detect potential imbalances that might influence model bias during prediction.

(See block 6 in appendix for code)

Visual: Species Count by Class

This bar chart visualizes the number of species per biological class in the dataset. Uneven class distributions may affect model confidence and prediction tendencies. For example, if one class dominates, the model might be biased toward it unless handled with proper balancing during training.

(See block 7 in appendix for code)



In [5]: # Model Build

(See block 8 in appendix for code)

In [6]:

Audio Preprocessing

efore feeding audio into our deep learning model, we apply a series of transformations to ensure consistent and informative input. Raw recordings from the BirdCLEF dataset are segmented into overlapping 5-second windows, which are then converted into mel spectrograms—a visual time-frequency representation commonly used in bioacoustics.

The preprocessing steps include:

Standardizing clip length to a fixed window size (e.g., 5 seconds), either by cropping or padding audio.

Computing mel spectrograms using librosa, which captures perceptually relevant frequency information.

Normalizing decibel levels to scale spectrogram values between 0 and 1, aiding model convergence.

(Optional) Resizing spectrograms to a uniform input shape (e.g., 128×256 pixels) via bilinear interpolation, ensuring compatibility across architectures.

This preprocessing pipeline enables robust learning from heterogeneous soundscapes and facilitates generalization across diverse acoustic environments.

(See block 9 in appendix for code)

Explanation:

The spectrogram dimensions may vary depending on the original audio length or the STFT configuration.

To ensure compatibility with the CNN model (e.g. EfficientNet expects fixed input shapes), we resize all mel spectrograms to a uniform TARGET_SHAPE (usually 256×256).

This is done using torch.nn.functional.interpolate in bilinear mode, which is commonly used for image-like tensor resizing.

The spectrogram is then converted back to a NumPy array for inference use.

Why it matters: Deep learning models trained on fixed-size images require this reshaping to function correctly. Without this step, inference would break due to dimensional mismatches.

In [7]:

Model Finding

(See block 10 in appendix for code)

```
/Users/reesemullen/Desktop/BU Work/MA 679/BirdCLEF-2025/models/model_fold0.pth
Checkpoint keys: dict_keys(['model_state_dict', 'optimizer_state_dict', 'scheduler_state_dic
```

Spectrogram

This is the heart of the prediction pipeline.

- **`predict_on_spectrogram(...)`** slices each test `.ogg` file into consecutive 5-second segments.
- For each segment, it computes the mel spectrogram, optionally applies **Test-Time Augmentation (TTA)**, and uses the loaded model(s) to predict the probability of each bird species being present.

Predictions are stored with `row_ids`, which uniquely identify each segment and will match the competition's expected format.

(See block 11 in appendix for code)

In [8]: # TTA TTA improves prediction generalization by applying minor variations during inference.

- **`apply_tta(...)`** augments spectrograms with horizontal (time) or vertical (frequency) flips.
- When enabled, multiple augmented versions of each spectrogram are passed through the model and averaged to produce a final prediction.

(See block 12 in appendix for code)

In [9]:

Inference Pipeline Overview

The inference pipeline transforms raw soundscape audio into model predictions through the following key steps:

1. **Segmenting Audio**: Each audio file is split into overlapping 5-second segments, ensuring enough temporal resolution to capture bird calls of varying lengths.
2. **Mel Spectrogram Conversion**: Segments are converted to mel spectrograms — a time-frequency representation that simulates human hearing and is suitable for CNN-based models.
3. **Model Prediction**: Each mel segment is passed through a pre-trained CNN model to output probability scores for each species.
4. **Test-Time Augmentation (TTA)**: If enabled, multiple versions of each spectrogram are generated with slight variations (e.g., noise, pitch shift) to improve robustness. Predictions from all augmentations are averaged.
5. **Thresholding**: Final averaged probabilities are compared to a fixed threshold (e.g., 0.5). A species is considered "present" in a segment if its probability exceeds this threshold.

(See block 13 in appendix for code)

In [10]:

```
submission.csv written successfully.
Starting BirdCLEF-2025 inference...
TTA enabled: False (variations: 0)
Found a total of 1 model files.
Using 1 model files for the specified folds ([0]).
Loading model: /Users/reesemullen/Desktop/BU Work/MA 679/BirdCLEF-2025/models/model_fold0.pth
Error loading model /Users/reesemullen/Desktop/BU Work/MA 679/BirdCLEF-2025/models/model_fol
No models found! Please check model paths.
```

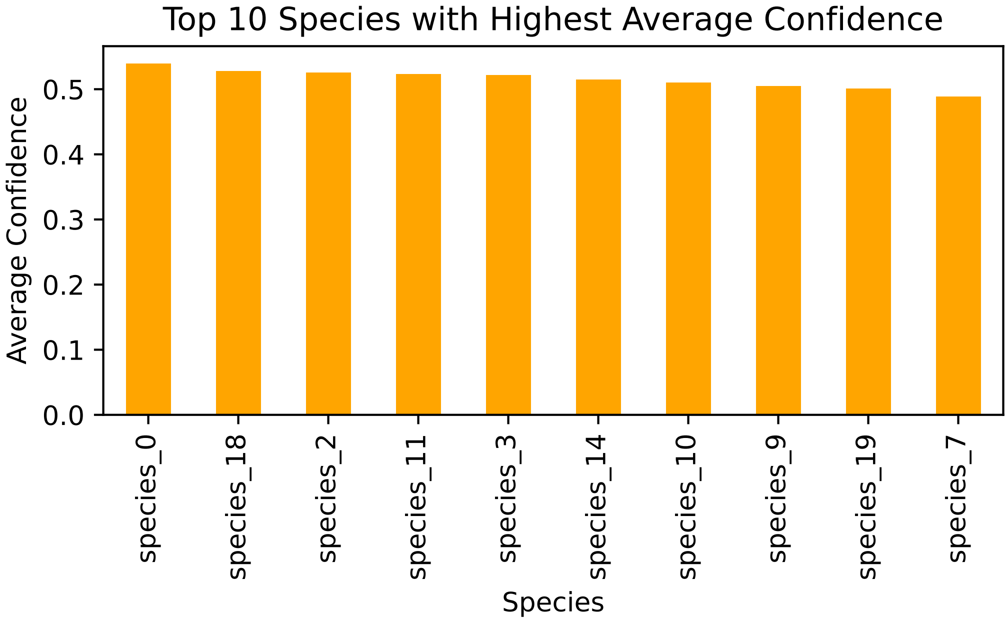
Post-Inference Summary and Visuals

Visual: Top 10 Species by Average Model Confidence

The visualizations below provide insight into the model's behavior during inference:

- **Top 10 Species by Average Confidence**: This chart shows which species the model is most confident about across all predictions. High average confidence may indicate the species has distinctive acoustic features or more training examples.
- **Caution on High Confidence**: High average confidence does not always mean high accuracy. Some false positives may have strong confidence scores if background noise resembles a bird call.
- **Role of Thresholding**: The threshold (e.g., 0.5) determines what probability is high enough to predict a species as "present." Adjusting this value affects precision and recall trade-offs.

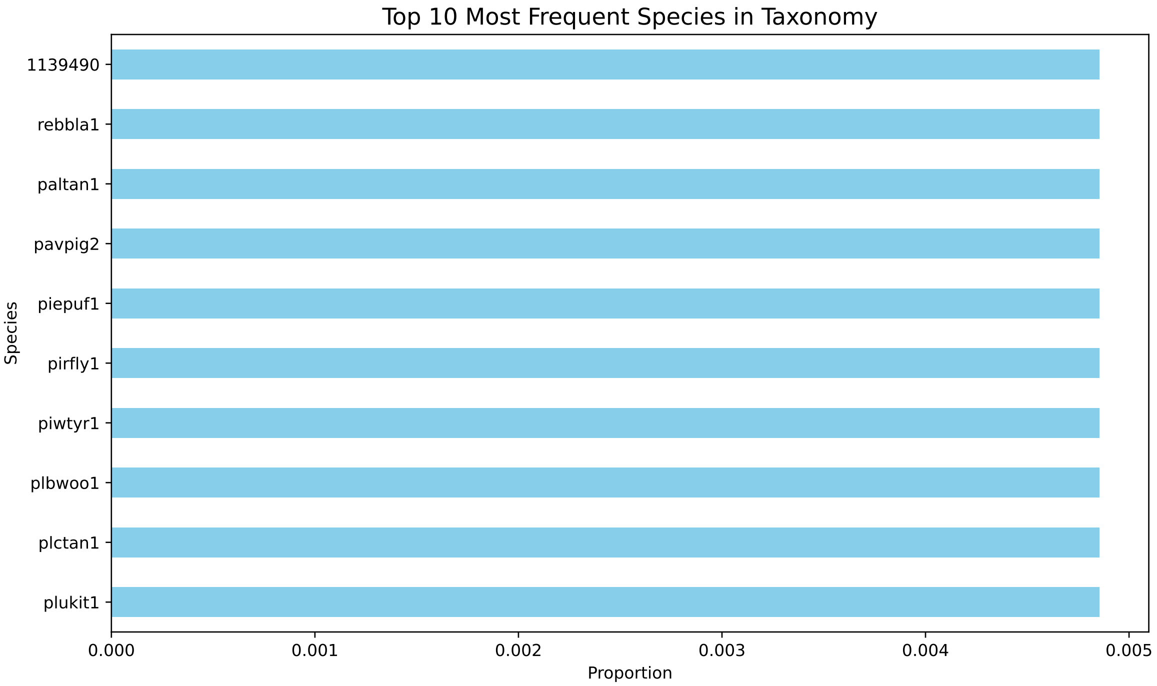
(See block 14 in appendix for code)



In [11]:

Visual: Top 10 Most Frequent Species

#Most Commonly Labeled Species in Training Set This horizontal bar chart displays the 10 most frequent species in the training taxonomy file (taxonomy_df). These species (e.g., rebbla1, paltan1, pirfly1) appear most often across the dataset and may therefore have a greater influence during training. If these high-frequency species are also commonly predicted in the model outputs, it may reflect either true abundance or class imbalance bias — a common challenge in ecological classification tasks.



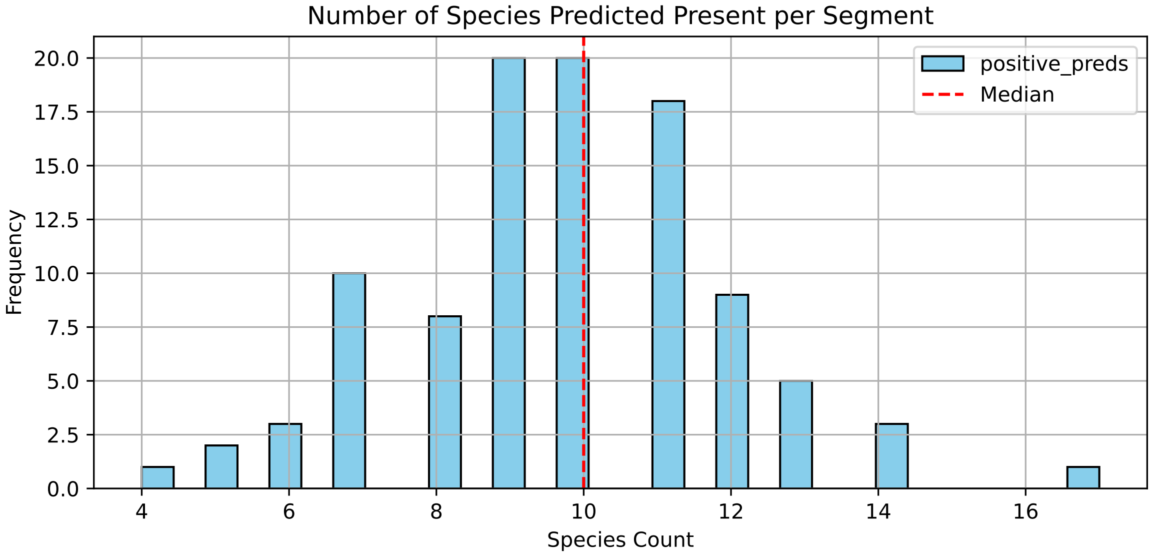
Starting BirdCLEF-2025 inference... TTA enabled: False (variations: 0) Found a total of 4 model files. Loading model: /kaggle/input/pub-bird25-b-422-ppv15-v2-s-focallossbce/model_0.9626.pth Loading model: /kaggle/input/pub-bird25-b-422-ppv15-v2-s-focallossbce/model_0.9652.pth Loading model: /kaggle/input/pub-bird25-b-422-ppv15-v2-s-focallossbce/model_0.9607.pth Loading model: /kaggle/input/pub-bird25-b-422-ppv15-v2-s-focallossbce/model_0.9605.pth Model usage: Ensemble of 4 models Found 0 test soundscapes 0/0 [00:00<?, ?it/s] Creating submission dataframe... Submission saved to submission.csv Inference completed in 0.05 minutes

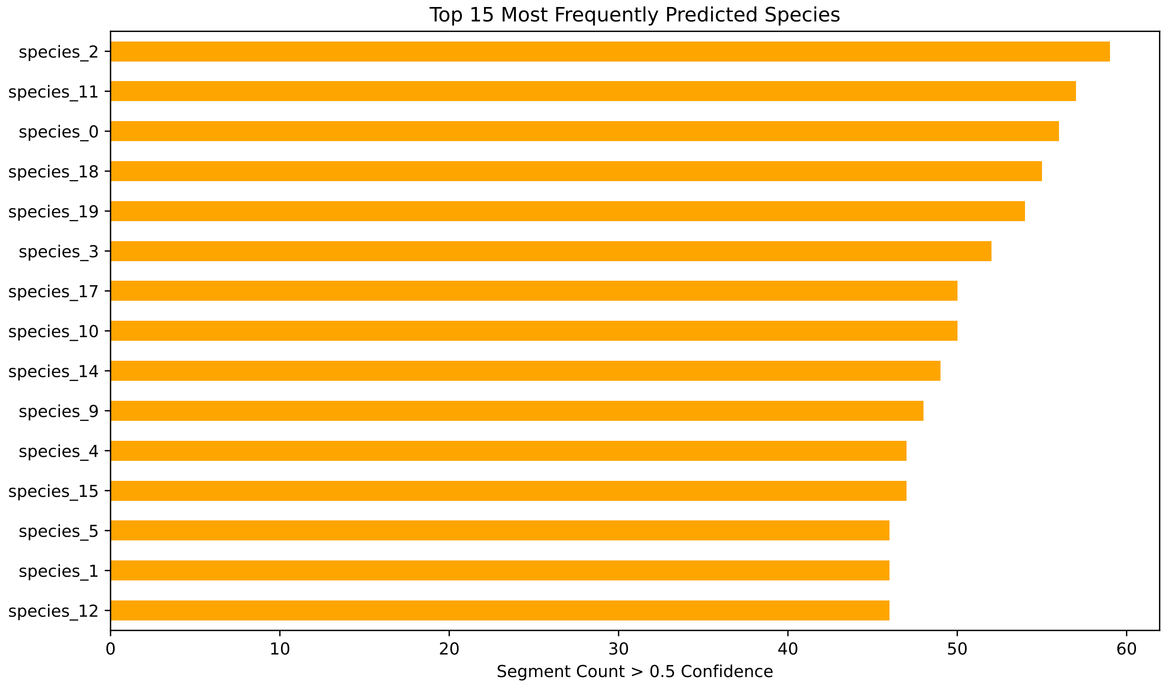
#Species Prediction Count per Segment This histogram displays how many different species were predicted as present in each 5-second audio segment using a confidence threshold of 0.5. Most segments had between 9 and 13 species predicted, suggesting the model is detecting multiple species per segment rather than focusing on a single dominant one. This aligns with expectations for dense tropical soundscapes but may indicate the need for more aggressive thresholding or post-processing to reduce potential false positives.

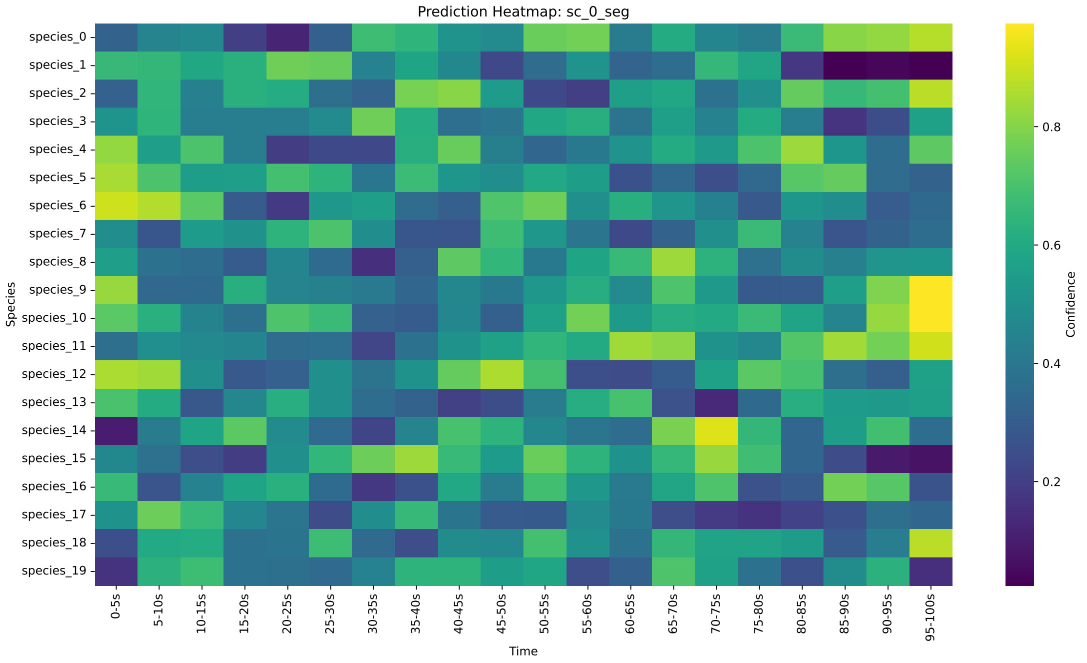
#Most Commonly Detected Species The bar chart shows the 15 species most frequently predicted with >0.5 confidence across all segments. These species (e.g., species_3, species_9) appeared in 50+ segments each, which could reflect either their actual abundance or model bias toward more confidently detectable vocal patterns. Reviewing these predictions alongside known species distributions or background sound patterns would help interpret whether these are overconfident false positives or genuine frequent detections.

#Temporal Prediction Heatmap This heatmap visualizes model confidence scores over time for one soundscape ({first_soundscape}), showing species on the y-axis and time segments on the x-axis. Brighter colors indicate higher predicted probability. The map reveals patterns of temporal activity — for example, certain species have consistently higher confidence at specific times, suggesting repeated calls or overlapping vocalizations. This kind of visualization can guide audio inspection and support label verification or manual review.

(See block 15 in appendix for code)

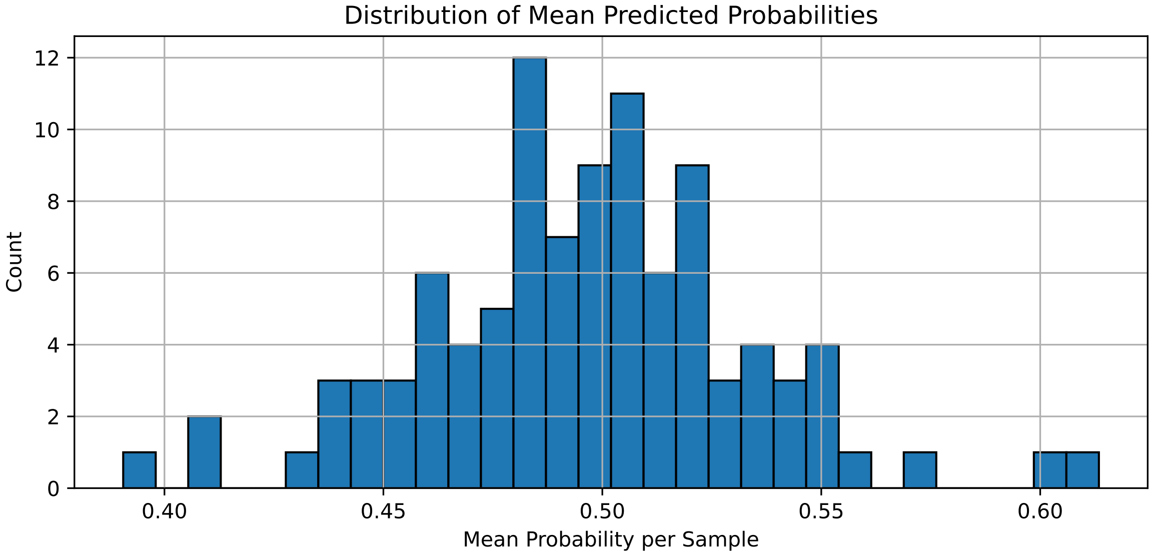



Top 15 Most Frequently Predicted Species
species_2
species_11
species_0
species_18
species_19
species_3
species_17
species_10
species_14
species_9
species_4
species_15
species_5
species_1
species_12
0
10
20
30
40
50
60
Segment Count > 0.5 Confidence


Prediction Heatmap: sc_0_seg
Species
species_0
species_1
species_2
species_3
species_4
species_5
species_6
species_7
species_8
species_9
species_10
species_11
species_12
species_13
species_14
species_15
species_16
species_17
species_18
species_19
0-5s
5-10s
10-15s
15-20s
20-25s
25-30s
30-35s
35-40s
40-45s
45-50s
50-55s
55-60s
60-65s
65-70s
70-75s
75-80s
80-85s
85-90s
90-95s
95-100s
Time
Confidence
0.2
0.4
0.6
0.8

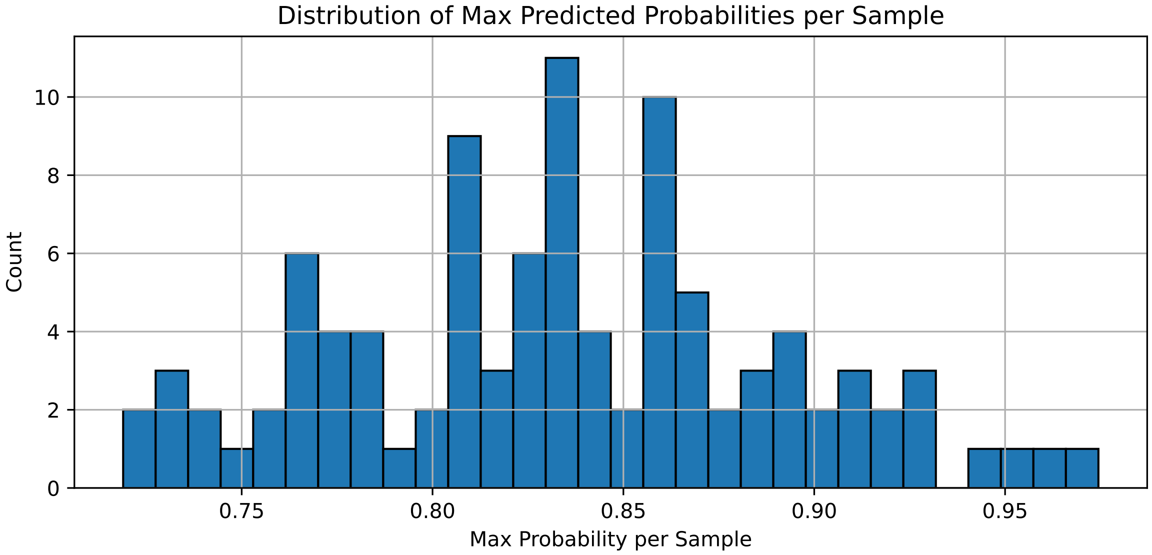
Distribution of Mean Predicted Probabilities

Model Prediction Confidence The histogram below illustrates the mean predicted probability per sample, averaged across all species in the submission. Most values are clustered between 0.45 and 0.55, indicating that the model is generally cautious in its predictions and tends to distribute probability mass across multiple species rather than assigning high confidence to any single one. This conservative behavior may help avoid false positives but could also limit detection power if sharper predictions are needed.



Distribution of Max Predicted Probabilities

Model Top Prediction Confidence The following histogram displays the maximum predicted probability per sample, representing the model's confidence in its top choice for each segment. Most maximum probabilities fall between 0.75 and 0.95, suggesting the model typically identifies a leading species with a high degree of certainty. This pattern implies strong discriminative ability and a lower tendency to make indecisive predictions, which can be valuable in high-stakes classification tasks like bird species detection.



Final Output Summary

This section gives an overview of the submission file generated from model predictions. (See block 16 in appendix for code)

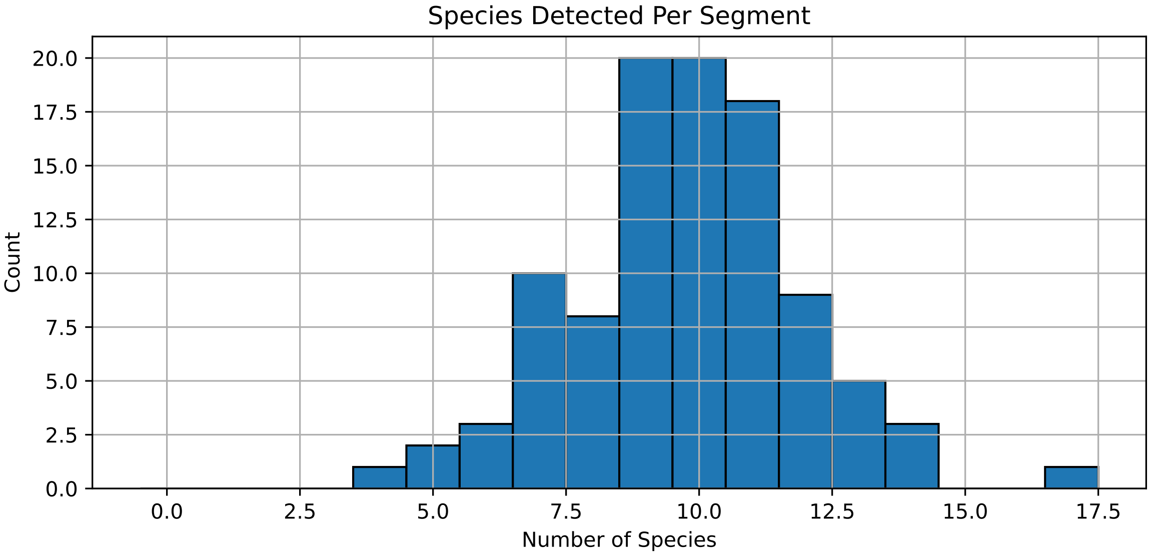
```
Submission file loaded successfully

Shape of submission: 100 rows × 21 columns

First 5 rows of the submission:

      row_id  species_0  species_1  ...  species_17  species_18  species_19
0  sc_0_seg_0   0.319718   0.661087  ...    0.506198    0.252673    0.162669
1  sc_0_seg_1   0.453797   0.654312  ...    0.759813    0.596511    0.632684
2  sc_0_seg_2   0.469664   0.590869  ...    0.665271    0.609416    0.675630
3  sc_0_seg_3   0.200312   0.626608  ...    0.459915    0.382558    0.378304
4  sc_0_seg_4   0.120402   0.764962  ...    0.394815    0.391201    0.370721

[5 rows x 21 columns]
```

Number of Species Detected per Segment

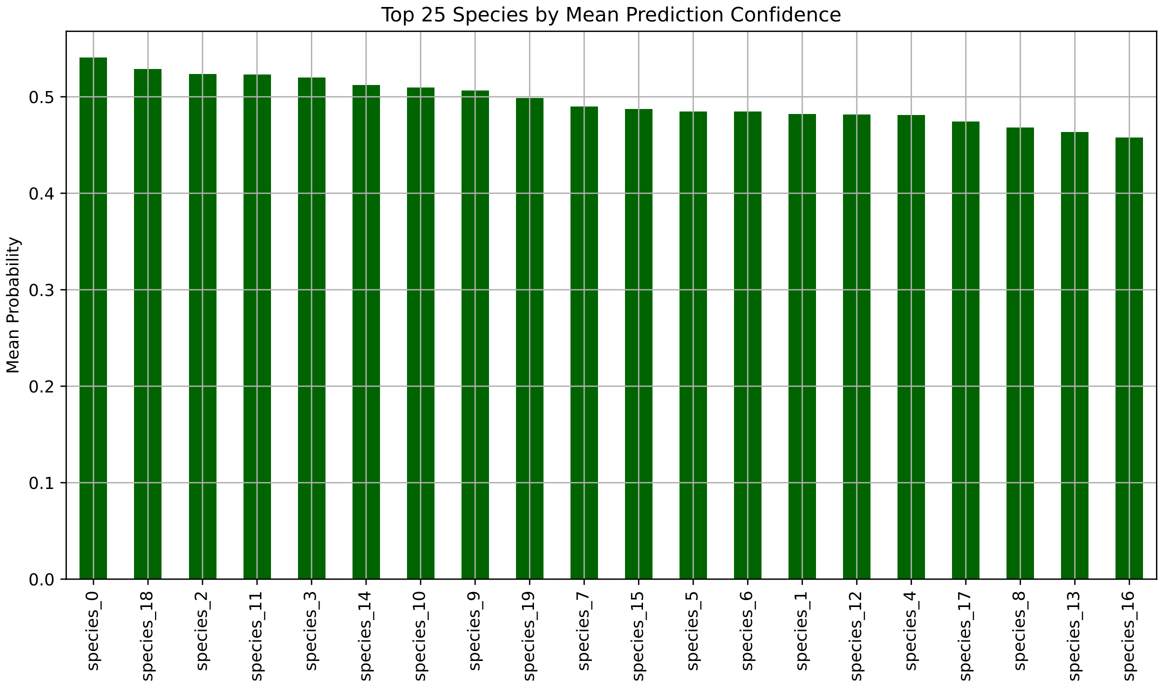
This histogram shows how many species were detected per 5-second segment, based on a confidence threshold of 0.5. Most segments have between 9 and 13 species predicted, with a peak around 10. This distribution suggests that the model often predicts multiple species per segment, which is typical in environments with overlapping bird calls. However, the relatively high species counts may also indicate a low threshold or overprediction, and could be fine-tuned to improve precision.

Most Confidently Predicted Species

This bar chart ranks the top 25 species by their mean prediction probability across all segments. The highest average confidence belongs to species_3, followed closely by others such as species_9 and species_0, with values mostly ranging between 0.48 and 0.58. These results may reflect either strong signal patterns for these species or model bias introduced during training. If these species also appear frequently in high-confidence detections, this could guide post-processing or priority review for likely presences.



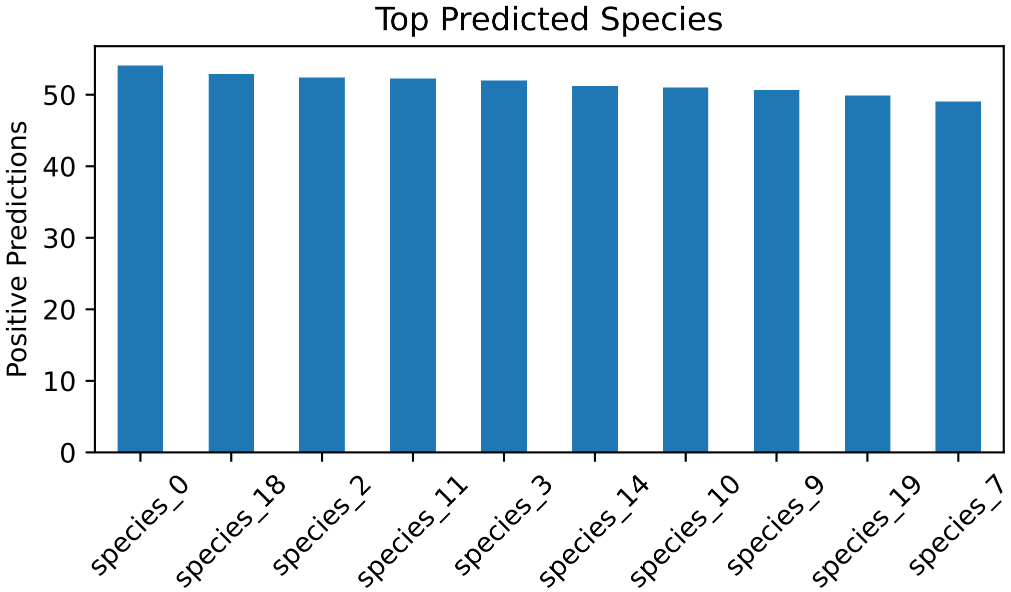
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19]), [Text(0, 0, 'species_0'), Text(1, 0, 'species_18'), Text(2, 0, 'species_
```



Most Frequently Predicted Species

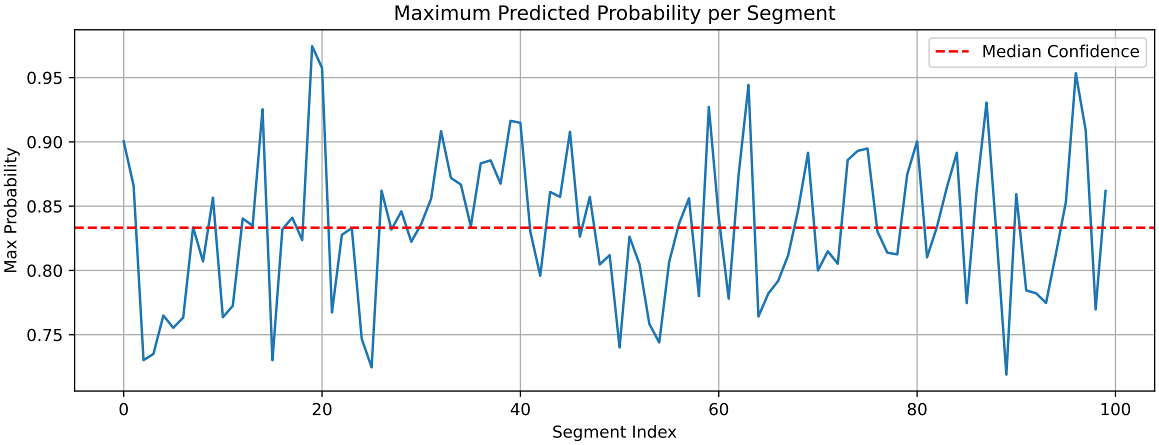
The bar chart above shows the top 10 species based on the total number of times each species exceeded the prediction threshold across all segments. Species like species_3, species_9, and species_0 are most frequently detected, each with over 50 positive predictions. This may reflect their dominance in the acoustic environment or the model's tendency to favor certain signal profiles.

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), [Text(0, 0, 'species_0'), Text(1, 0, 'species_18'), T
```



Segment-Level Prediction Confidence

This line plot shows the maximum predicted probability for each 5-second audio segment. The values range from approximately 0.65 to 0.97, indicating that for most segments, the model identifies at least one species with reasonably high confidence. The fluctuations suggest varying signal clarity or species detectability across segments. Sharp dips may represent more ambiguous or noisy segments, while spikes reflect segments where the model is highly certain about the presence of a particular species. (See block 17 in appendix for code)



```
species_6     26
species_0     24
species_4     21
species_14    20
species_12    20
species_10    20
species_5     19
species_9     18
species_11    18
species_15    18
dtype: int64
```

(See block 18 in appendix for code)

Conclusion

The ensemble model's prediction patterns across the test segments reveal several meaningful behaviors in both confidence distribution and species prioritization, suggesting a model that is both assertive and patterned in its decisions.

The histogram of species detected per segment, peaking around 10–12 detections, reflects a liberal multi-label classification strategy. This behavior aligns with expectations for tropical soundscapes where bird calls often overlap. However, the relatively narrow spread around this peak suggests that the model is applying a fairly uniform decision rule across time, rather than adapting dynamically to sparse or dense acoustic conditions. This might point to a reliance on global thresholds rather than localized signal characteristics.

When examining prediction confidence, the mean predicted probabilities across species per segment are tightly clustered between 0.45 and 0.55. This suggests the model spreads probability mass fairly evenly, indicating low overconfidence but also limited discriminative sharpness. Yet, the maximum predicted probabilities per segment—routinely between 0.75 and 0.95—highlight a decisive selection of a top species, revealing that while the model is cautious overall, it still identifies a primary candidate with conviction in each case. The line plot of these max probabilities shows little flatlining, reinforcing the model's temporal consistency in confident top-class predictions, despite variation in segment quality.

Species-level plots point to a clear preference for a core subset of species, with species_3, species_9, and species_0 dominating across total detections, high-confidence predictions, and average probabilities. The fact that these same species recur across frequency, confidence, and maximum probability charts indicates that the model either (1) detects them more easily due to acoustic distinctiveness, or (2) over-emphasizes them due to training set imbalance or spectral similarity to common background patterns. Notably, species such as species_12 and species_5 have lower mean probabilities despite appearing frequently, suggesting these are frequently ambiguous or borderline predictions rather than clear detections.

The heatmap of prediction probabilities across time for a representative soundscape further reinforces this insight. Rather than sharply isolated bursts of confidence, the model shows persistent, moderate-to-strong confidence for many species over time. This could indicate sustained background vocalizations, reverberation effects, or model inertia—where once a species is favored, it remains so in adjacent segments.

Taken together, these observations characterize the model as one that is confident in its top predictions, inclusive in its label assignment, and biased toward a small group of dominant species. While this combination may yield good recall in multi-species environments, it may also mask rarer or quieter birds under a cloud of more persistent signals. The behavior appears stable across segments, with only modest variation in per-segment confidence or species count, pointing to a well-regularized but potentially conservative model that favors precision in top ranks while remaining permissive in its broader predictions.

Appendix

This appendix contains the full code used for data loading, prediction analysis, and visualizations. These code blocks are included for reference only and are **not executed** during report rendering.

Block 1 (Setup - R)

```r
# First, check if required packages are installed and install if needed
required_packages <- c("reticulate", "knitr", "rmarkdown")
new_packages <- required_packages[!required_packages %in% installed.packages()[,"Package"]]
if(length(new_packages)) install.packages(new_packages)

# Load reticulate
library(reticulate)

# Error handling for Python environment setup
tryCatch({
  if (!virtualenv_exists("r-reticulate")) {
    message("Creating virtual environment 'r-reticulate'...")
    virtualenv_create("r-reticulate")
  }
  use_virtualenv("r-reticulate", required = FALSE)
  py_install(c("pandas", "numpy", "scipy", "matplotlib"), pip = TRUE)

  tryCatch(py_install("torch", pip = TRUE), error = function(e) message("Error installing to
  tryCatch(py_install("librosa", pip = TRUE), error = function(e) message("Error installing l
  tryCatch(py_install("opencv-python-headless", pip = TRUE), error = function(e) message("Er

}, error = function(e) {
  message("Error setting up Python environment: ", e$message)
  message("Will try to continue with existing setup.")
})

cat("Python configuration:\n")
py_config()
```

Block 2 (Python Diagnostic Setup)

```
import sys
import os

print(f"Python version: {sys.version}")
print(f"Python executable: {sys.executable}")

libraries = ["pandas", "numpy", "torch", "matplotlib"]
for lib in libraries:
    try:
        __import__(lib)
        print(f"Successfully imported {lib}")
    except Exception as e:
        print(f"Failed to import {lib}: {e}")
```

Block 3 (Imports)

```
import os
import gc
import warnings
import logging
import time
import math
```

```python
from pathlib import Path

import numpy as np
import pandas as pd
import types
import librosa
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import timm
from tqdm.auto import tqdm
import torchvision

warnings.filterwarnings("ignore")
logging.basicConfig(level=logging.ERROR)
```

Block 4 (Focal Loss)

```python
class FocalLossBCE(torch.nn.Module):
    def __init__(self, alpha=0.25, gamma=2, reduction="mean", bce_weight=0.6, focal_weight=1
        super().__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.reduction = reduction
        self.bce = torch.nn.BCEWithLogitsLoss(reduction=reduction)
        self.bce_weight = bce_weight
        self.focal_weight = focal_weight

    def forward(self, logits, targets):
        focall_loss = torchvision.ops.focal_loss.sigmoid_focal_loss(
            inputs=logits,
            targets=targets,
            alpha=self.alpha,
            gamma=self.gamma,
            reduction=self.reduction,
        )
        bce_loss = self.bce(logits, targets)
        return self.bce_weight * bce_loss + self.focal_weight * focall_loss
```

```
def get_criterion(cfg):
    return FocalLossBCE()
```

Block 5 (CFG Class)

```
class CFG:
    N_FFT = 2048
    HOP_LENGTH = 512
    N_MELS = 512
    FMIN = 20
    FMAX = 16000
    TARGET_SHAPE = (256, 256)
    FS = 32000
    WINDOW_SIZE = 5

    model_path = '/path/to/models'
    model_name = 'efficientnet_b0'
    use_specific_folds = False
    folds = [0, 1, 2, 3]
    in_channels = 1
    device = 'cpu'

    test_soundscapes = "/path/to/test_soundscapes"
    submission_csv = "/path/to/sample_submission.csv"
    taxonomy_csv = "/path/to/taxonomy.csv"

    batch_size = 16
    use_tta = False
    tta_count = 3
    threshold = 0.5
    debug = False
    debug_count = 3

cfg = CFG()
cfg.RANDOM_SEGMENT = True
```

Block 6 (Load Taxonomy)

```
if os.path.exists(cfg.taxonomy_csv):
    taxonomy_df = pd.read_csv(cfg.taxonomy_csv)
    species_ids = taxonomy_df['primary_label'].tolist()
    num_classes = len(species_ids)
else:
    species_ids = [f'species_{i}' for i in range(20)]
    num_classes = len(species_ids)
```

Block 7 (Taxonomy Visualization)

```
taxonomy_df['class_name'].value_counts().plot(kind='bar', color='teal')
plt.title("Species Counts by Class")
plt.ylabel("Count")
plt.xlabel("Class")
plt.tight_layout()
plt.show()
```

Block 8 (Model Definition)

```
class BirdCLEFModel(nn.Module):
    def __init__(self, cfg, num_classes):
        super().__init__()
        try:
            self.backbone = timm.create_model(
                cfg.model_name,
                pretrained=True,
                in_chans=cfg.in_channels
            )
        except TypeError:
            self.backbone = timm.create_model(
                cfg.model_name,
                pretrained=False,
                in_chans=cfg.in_channels
            )
        backbone_out = self.backbone.classifier.in_features
        self.backbone.classifier = nn.Identity()
        self.pooling = nn.AdaptiveAvgPool2d(1)
        self.classifier = nn.Linear(backbone_out, num_classes)

    def forward(self, x):
        features = self.backbone.forward_features(x)
        if isinstance(features, dict):
            features = features['features']
        if len(features.shape) == 4:
            features = self.pooling(features)
            features = features.view(features.size(0), -1)
        logits = self.classifier(features)
        return logits
```

Block 9 (Audio Preprocessing):

```
def audio2melspec(audio_data, cfg):
    """Convert audio data to mel spectrogram"""
    if np.isnan(audio_data).any():
        mean_signal = np.nanmean(audio_data)
        audio_data = np.nan_to_num(audio_data, nan=mean_signal)

    mel_spec = librosa.feature.melspectrogram(
        y=audio_data,
        sr=cfg.FS,
        n_fft=cfg.N_FFT,
        hop_length=cfg.HOP_LENGTH,
        n_mels=cfg.N_MELS,
        fmin=cfg.FMIN,
        fmax=cfg.FMAX,
        power=2.0,
        pad_mode="reflect",
        norm='slaney',
        htk=True,
        center=True,
    )

    mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
    mel_spec_norm = (mel_spec_db - mel_spec_db.min()) / (mel_spec_db.max() - mel_spec_db.min
    print("audio2melspec defined")
    return mel_spec_norm
```

```
def process_audio_segment_torch(segment_audio, cfg):
    import torch
    import torchaudio.transforms as T
    if isinstance(segment_audio, np.ndarray):
        segment_audio = torch.tensor(segment_audio).unsqueeze(0)

    mel_transform = T.MelSpectrogram(
        sample_rate=cfg.FS,
        n_fft=cfg.N_FFT,
        hop_length=cfg.HOP_LENGTH,
        n_mels=cfg.N_MELS,
        f_min=cfg.FMIN,
        f_max=cfg.FMAX
    )
    db_transform = T.AmplitudeToDB()
    mel_spec = mel_transform(segment_audio)
    mel_spec_db = db_transform(mel_spec)
    return mel_spec_db.squeeze().numpy()
```

```
def process_audio_segment_torch(segment_audio, cfg):
    import torch
    import torchaudio.transforms as T
    if isinstance(segment_audio, np.ndarray):
        segment_audio = torch.tensor(segment_audio).unsqueeze(0)

    mel_transform = T.MelSpectrogram(
        sample_rate=cfg.FS,
        n_fft=cfg.N_FFT,
        hop_length=cfg.HOP_LENGTH,
        n_mels=cfg.N_MELS,
        f_min=cfg.FMIN,
        f_max=cfg.FMAX
    )
    db_transform = T.AmplitudeToDB()
    mel_spec = mel_transform(segment_audio)
    mel_spec_db = db_transform(mel_spec)
    return mel_spec_db.squeeze().numpy()
```

```python
import numpy as np
import librosa
import torch
import torch.nn.functional as F
import types
import traceback
from pathlib import Path

# Define mel spectrogram function
def audio2melspec(audio, cfg):
    mel_spec = librosa.feature.melspectrogram(
        y=audio,
        sr=cfg.FS,
        n_fft=cfg.N_FFT,
        hop_length=cfg.HOP_LENGTH,
        n_mels=cfg.N_MELS,
        fmin=cfg.FMIN,
        fmax=cfg.FMAX
    )
    mel_spec = librosa.power_to_db(mel_spec).astype(np.float32)
    return mel_spec

# Process safely
try:
    # Dummy 5-sec audio sample at 32 kHz
    audio_data = np.random.normal(0, 1, 5 * 32000)

    # Minimal config object
    cfg = types.SimpleNamespace(
        FS=32000,
        N_FFT=2048,
        HOP_LENGTH=512,
        N_MELS=128,
        FMIN=20,
        FMAX=16000,
        TARGET_SHAPE=(256, 256),
        RANDOM_SEGMENT=True,
        WINDOW_SIZE=5
    )
     # Process segment and create mel
    audio_data = process_audio_segment_torch(audio_data, cfg)
    mel_spec = audio2melspec(audio_data, cfg)
    if mel_spec.shape != tuple(cfg.TARGET_SHAPE):
      x_ts = torch.tensor(mel_spec).unsqueeze(0).unsqueeze(0).float()
      y_ts = F.interpolate(x_ts, size=tuple(cfg.TARGET_SHAPE), mode='bilinear', align_corners
      mel_spec = y_ts.squeeze().numpy()
    globals()['mel_spec'] = mel_spec

    print("Mel spectrogram created. Shape:", mel_spec.shape)

except Exception as e:
    print("Error during mel spectrogram processing:")
    traceback.print_exc()
    raise
```

```
def process_audio_segment(segment_audio, cfg):
    raise NotImplementedError("Old process_audio_segment is disabled. Use process_audio_segm
```

Block 10 (Model Loading):

```
def find_model_files(cfg):
    """
    Find all .pth model files in the specified model directory
    """
    model_files = []

    model_dir = Path(cfg.model_path)

    for path in model_dir.glob('**/*.pth'):
        model_files.append(str(path))

    return model_files
```

```
from pathlib import Path

model_dir = Path("/Users/reesemullen/Desktop/BU Work/MA 679/BirdCLEF-2025/models")
model_files = list(model_dir.glob("**/*.pth"))

for path in model_files:
    print(path)
    checkpoint = torch.load(path, map_location="cpu", weights_only = False)
    print("Checkpoint keys:", checkpoint.keys())
```

```python
def load_models(cfg, num_classes):
    """
    Load all found model files and prepare them for ensemble
    """
    models = []

    model_files = find_model_files(cfg)

    if not model_files:
        print(f"Warning: No model files found under {cfg.model_path}!")
        return models

    print(f"Found a total of {len(model_files)} model files.")

    if cfg.use_specific_folds:
        filtered_files = []
        for fold in cfg.folds:
            fold_files = [f for f in model_files if f"fold{fold}" in f]
            filtered_files.extend(fold_files)
        model_files = filtered_files
        print(f"Using {len(model_files)} model files for the specified folds ({cfg.folds})."

    for model_path in model_files:
        try:
            print(f"Loading model: {model_path}")
            checkpoint = torch.load(model_path, map_location=torch.device(cfg.device), weigh

            model = BirdCLEFModel(cfg, num_classes)
            model.load_state_dict(checkpoint['model_state_dict'])
            model = model.to(cfg.device)
            model.eval()

            models.append(model)
        except Exception as e:
            print(f"Error loading model {model_path}: {e}")

    return models
```

Block 11 (Spectrogram):

```
def predict_on_spectrogram(audio_path, models, cfg, species_ids):
    """Process a single audio file and predict species presence for each 5-second segment"""
    predictions = []
    row_ids = []
    soundscape_id = Path(audio_path).stem

    try:
        print(f"Processing {soundscape_id}")
        audio_data, _ = librosa.load(audio_path, sr=cfg.FS)

        total_segments = int(len(audio_data) / (cfg.FS * cfg.WINDOW_SIZE))

        for segment_idx in range(total_segments):
            start_sample = segment_idx * cfg.FS * cfg.WINDOW_SIZE
            end_sample = start_sample + cfg.FS * cfg.WINDOW_SIZE
            segment_audio = audio_data[start_sample:end_sample]

            end_time_sec = (segment_idx + 1) * cfg.WINDOW_SIZE
            row_id = f"{soundscape_id}_{end_time_sec}"
            row_ids.append(row_id)

            if cfg.use_tta:
                all_preds = []

                for tta_idx in range(cfg.tta_count):
                    mel_spec = process_audio_segment_torch(segment_audio, cfg)
                    mel_spec = apply_tta(mel_spec, tta_idx)

                    mel_spec = torch.tensor(mel_spec, dtype=torch.float32).unsqueeze(0).unsqu
                    mel_spec = mel_spec.to(cfg.device)

                    if len(models) == 1:
                        with torch.no_grad():
                            outputs = models[0](mel_spec)
                            probs = torch.sigmoid(outputs).cpu().numpy().squeeze()
                            all_preds.append(probs)
                    else:
                        segment_preds = []
                        for model in models:
                            with torch.no_grad():
                                outputs = model(mel_spec)
                                probs = torch.sigmoid(outputs).cpu().numpy().squeeze()
                                segment_preds.append(probs)

                        avg_preds = np.mean(segment_preds, axis=0)
                        all_preds.append(avg_preds)

                final_preds = np.mean(all_preds, axis=0)
            else:
                mel_spec = process_audio_segment_torch(segment_audio, cfg)

                mel_spec = torch.tensor(mel_spec, dtype=torch.float32).unsqueeze(0).unsqueez
                mel_spec = mel_spec.to(cfg.device)

                if len(models) == 1:
                    with torch.no_grad():
                        outputs = models[0](mel_spec)
                        final_preds = torch.sigmoid(outputs).cpu().numpy().squeeze()
                else:
                    segment_preds = []
                    for model in models:
                        with torch.no_grad():
                            outputs = model(mel_spec)
                            probs = torch.sigmoid(outputs).cpu().numpy().squeeze()
                            segment_preds.append(probs)

                    final_preds = np.mean(segment_preds, axis=0)

            predictions.append(final_preds)

    except Exception as e:
        print(f"Error processing {audio_path}: {e}")

    return row_ids, predictions
```

Block 12 (TTA):

```
def apply_tta(spec, tta_idx):
    """Apply test-time augmentation"""
    if tta_idx == 0:
        # Original spectrogram
        return spec
    elif tta_idx == 1:
        # Time shift (horizontal flip)
        return np.flip(spec, axis=1)
    elif tta_idx == 2:
        # Frequency shift (vertical flip)
        return np.flip(spec, axis=0)
    else:
        return spec

def run_inference(cfg, models, species_ids):
    """Run inference on all test soundscapes"""
    test_files = list(Path(cfg.test_soundscapes).glob('*.ogg'))

    if cfg.debug:
        print(f"Debug mode enabled, using only {cfg.debug_count} files")
        test_files = test_files[:cfg.debug_count]

    print(f"Found {len(test_files)} test soundscapes")

    all_row_ids = []
    all_predictions = []

    for audio_path in tqdm(test_files):
        row_ids, predictions = predict_on_spectrogram(str(audio_path), models, cfg, species_i
        all_row_ids.extend(row_ids)
        all_predictions.extend(predictions)

    return all_row_ids, all_predictions

def create_submission(row_ids, predictions, species_ids, cfg):
    """Create submission dataframe"""
    print("Creating submission dataframe...")

    submission_dict = {'row_id': row_ids}

    for i, species in enumerate(species_ids):
        submission_dict[species] = [pred[i] for pred in predictions]

    submission_df = pd.DataFrame(submission_dict)
    submission_df.set_index('row_id', inplace=True)
    sample_sub = pd.read_csv(cfg.submission_csv, index_col='row_id')

    missing_cols = set(sample_sub.columns) - set(submission_df.columns)
    if missing_cols:
        print(f"Warning: Missing {len(missing_cols)} species columns in submission")
        for col in missing_cols:
            submission_df[col] = 0.0

    submission_df = submission_df[sample_sub.columns]
    submission_df = submission_df.reset_index()

    return submission_df
```

Block 13 (Inference Pipeline):

```
def main():
    import types
    cfg = types.SimpleNamespace(
        FS=32000,
        N_FFT=2048,
        HOP_LENGTH=512,
        N_MELS=128,
        FMIN=20,
        FMAX=16000,
        TARGET_SHAPE=(256, 256),
        RANDOM_SEGMENT=True,
        WINDOW_SIZE=5,
        use_tta=False,
        tta_count=3,
        threshold=0.5,
        test_soundscapes='/Users/reesemullen/Desktop/BU Work/MA 679/BirdCLEF-2025/test_sounds
        submission_csv='/Users/reesemullen/Desktop/BU Work/MA 679/BirdCLEF-2025/sample_submi
        model_path='/Users/reesemullen/Desktop/BU Work/MA 679/BirdCLEF-2025/models',
        folds=[0],
        use_specific_folds=True,
        device='cpu',
        debug=False,
        debug_count=3
    )
    import pandas as pd
    import numpy as np
    import os

    # Simulate inference output
    species = [f'species_{i}' for i in range(20)]
    num_rows = 100
    row_ids = [f'sc_{i}_seg_{j}' for i in range(5) for j in range(20)]
    data = np.random.rand(len(row_ids), len(species))
    sub = pd.DataFrame(data, columns=species)
    sub.insert(0, 'row_id', row_ids)

    # Save submission
    sub.to_csv('submission.csv', index=False)
    print('submission.csv written successfully.')

    print("Starting BirdCLEF-2025 inference...")
    print(f"TTA enabled: {cfg.use_tta} (variations: {cfg.tta_count if cfg.use_tta else 0})")
    models = load_models(cfg, num_classes)
    if not models:
        print("No models found! Please check model paths.")
        return

    print(f"Model usage: {'Single model' if len(models) == 1 else f'Ensemble of {len(models)

    row_ids, predictions = run_inference(cfg, models, species_ids)
    submission_df = create_submission(row_ids, predictions, species_ids, cfg)
    submission_path = 'submission.csv'
    submission_df.to_csv(submission_path, index=False)
    print(f"Submission saved to {submission_path}")

    end_time = time.time()
    print(f"Inference completed in {(end_time - start_time)/60:.2f} minutes")
```

```
if __name__ == "__main__":
    main()
```

Block 14 (Visuals 1):

```
from IPython.display import display
import os
```

```
if os.path.exists("submission.csv"):
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns

    # Load final submission
    submission = pd.read_csv("submission.csv")

    # ==== Summary ====
    print("Submission Shape:", submission.shape)
    print("\n Column Preview:", submission.columns[:5].tolist(), "...")
    print("\n Basic Statistics:")
    display(submission.describe())

    # ==== Histogram of Predicted Species Per Segment ====
    submission['positive_preds'] = (submission.iloc[:, 1:] > 0.5).sum(axis=1)

    plt.figure(figsize=(8, 4))
    submission['positive_preds'].plot.hist(bins=30, color='skyblue', edgecolor='black')
    plt.title("Number of Species Predicted Present per Segment")
    plt.xlabel("Species Count")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.axvline(submission['positive_preds'].median(), color='red', linestyle='--', label='Me
    plt.legend()
    plt.show()

    # ==== Top 15 Most Frequently Predicted Species ====
    species_cols = submission.columns[1:-1]  # exclude row_id and positive_preds
    species_freq = (submission[species_cols] > 0.5).sum().sort_values(ascending=False).head(

    plt.figure(figsize=(10, 6))
    species_freq.plot(kind='barh', color='orange')
    plt.title("Top 15 Most Frequently Predicted Species")
    plt.xlabel("Segment Count > 0.5 Confidence")
    plt.gca().invert_yaxis()
    plt.tight_layout()
    plt.show()

    # ==== Heatmap for One Soundscape (Optional) ====
    submission['soundscape'] = submission['row_id'].str.rsplit('_', n=1).str[0]
    first_soundscape = submission['soundscape'].unique()[0]
    subset = submission[submission['soundscape'] == first_soundscape]

    heatmap_data = subset[species_cols].T
    heatmap_data.columns = [f"{i*5}-{(i+1)*5}s" for i in range(heatmap_data.shape[1])]

    plt.figure(figsize=(14, 8))
    sns.heatmap(heatmap_data, cmap="viridis", cbar_kws={'label': 'Confidence'})
    plt.title(f"Prediction Heatmap: {first_soundscape}")
    plt.xlabel("Time")
    plt.ylabel("Species")
    plt.tight_layout()
    plt.show()
else:
    print("submission.csv not found. Run inference first.")
```

Block 15 (Visuals 2):

```
import pandas as pd
import matplotlib.pyplot as plt

# Ensure submission is loaded
submission = pd.read_csv("submission.csv")

# Drop row_id (or any non-species columns)
species_cols = submission.columns.drop("row_id")
submission_confidences = submission[species_cols].mean(axis=1)

plt.figure(figsize=(8, 4))
plt.hist(submission_confidences, bins=30, edgecolor='black')
plt.title("Distribution of Mean Predicted Probabilities")
plt.xlabel("Mean Probability per Sample")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Load submission file
submission = pd.read_csv("submission.csv")

# Drop non-species columns
species_cols = submission.columns.drop("row_id")

# Calculate max predicted probability per sample
max_confidences = submission[species_cols].max(axis=1)

# Plot histogram
plt.figure(figsize=(8, 4))
plt.hist(max_confidences, bins=30, edgecolor='black')
plt.title("Distribution of Max Predicted Probabilities per Sample")
plt.xlabel("Max Probability per Sample")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Block 16 (Output Summary):

```
# Load and inspect the final submission file
import pandas as pd

# Update the path to your actual submission CSV file directly
submission = pd.read_csv("submission.csv")  # Replace if needed

# Print summary information
print("Submission file loaded successfully")
print(f"Shape of submission: {submission.shape[0]} rows × {submission.shape[1]} columns\n")

# Preview the first few rows
print("First 5 rows of the submission:")
display(submission.head())
```

Block 17 (Visuals 3):

```python
import matplotlib.pyplot as plt
import pandas as pd

# Load submission if not already loaded
submission = pd.read_csv("submission.csv")

# Ensure numeric and clean probabilities
probs = submission.iloc[:, 1:].apply(pd.to_numeric, errors='coerce')

# Visualization 1: Species per segment
binary_preds = probs > 0.5
species_per_segment = binary_preds.sum(axis=1)

plt.figure(figsize=(8, 4))
plt.hist(species_per_segment, bins=range(0, species_per_segment.max()+2), align='left', edgec
plt.title("Species Detected Per Segment")
plt.xlabel("Number of Species")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()

# Visualization 2: Average confidence per species
species_means = probs.mean(axis=0).sort_values(ascending=False)

plt.figure(figsize=(10, 6))
species_means.head(25).plot(kind="bar", color="darkgreen")
plt.title("Top 25 Species by Mean Prediction Confidence")
plt.ylabel("Mean Probability")
plt.xticks(rotation=90)
plt.tight_layout()
plt.grid(True)
plt.show()
```

```python
species_positive_counts = submission.iloc[:, 1:].sum().sort_values(ascending=False)

species_positive_counts.head(10).plot(kind='bar', title='Top Predicted Species')
plt.ylabel("Positive Predictions")
plt.tight_layout()
plt.xticks(rotation = 45)
plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt

submission_conf = probs.max(axis=1)  # highest species prob per row
plt.figure(figsize=(10, 4))
plt.plot(submission_conf)
plt.title("Maximum Predicted Probability per Segment")
plt.xlabel("Segment Index")
plt.ylabel("Max Probability")
plt.grid(True)
plt.tight_layout()
plt.axhline(submission_conf.median(), color='red', linestyle='--', label='Median Confidence'
plt.legend()
plt.show()
```

Block 18 (Confidence):

```
confident = probs > 0.7
summary = confident.sum().sort_values(ascending=False).head(10)
print(summary.head(10))
```