Mamethierno Gadiaga
Richard Cenedella
University Of Rhode Island
CSC_561: Neural Networks & Deep Learning

Title: Comprehensive Strategy for a High-Performing `BirdCLEF+ 2025 Ensemble

Introduction:

For the final project in this class (CSC_561 Neural Networks & Deep Learning), we decided to join this kaggle competition: https://www.kaggle.com/competitions/birdclef-2025. The main objective of this project is to be able to analyze audio recordings of nature to identify the many different species of animals found in the El Silencio Natural reserve in Colombia. The identification will be based on the unique vocalizations that each animal makes. The types of animals that are represented in this kaggle competition's dataset include various species of birds, amphibians, mammals and insects. This data was captured and collected in a couple different ways. The first way is through user uploaded submissions to animal identification websites. Another way it was collected was by passive acoustic monitoring devices that were placed in the Magdalena valley of Colombia. The end goal of this project will be to enhance the biodiversity monitoring efforts for the researchers in the nature reserve. They want to be able to automate the detection and classification of different taxonomic groups found in the area. This research stems from their ecological restoration efforts in the region. Historically this area was a humid tropical rainforest and it was a significant biodiversity hotspot. While it is still a biodiversity hotspot, the rainforest has been ravaged by illegal logging operations, farming and cattle ranching over time. Today researchers aim to restore the forest and wetlands through community outreach and other biodiversity monitoring programs. Their goals are to increase endangered animal populations back to healthy population numbers. This requires some form of monitoring in order to be able to track population numbers in the wild. Currently, biodiversity surveys are done manually by people on the ground monitoring groups of animals. This is costly and labor intensive for the researchers. By employing machine learning techniques, they hope to be able to automate their analysis of passive acoustic monitoring data and employ greater temporal resolution than ever before. This will provide them with new insights into their restoration efforts and help them to improve the biodiversity in the area and return the forest back to its natural and sustainable state.

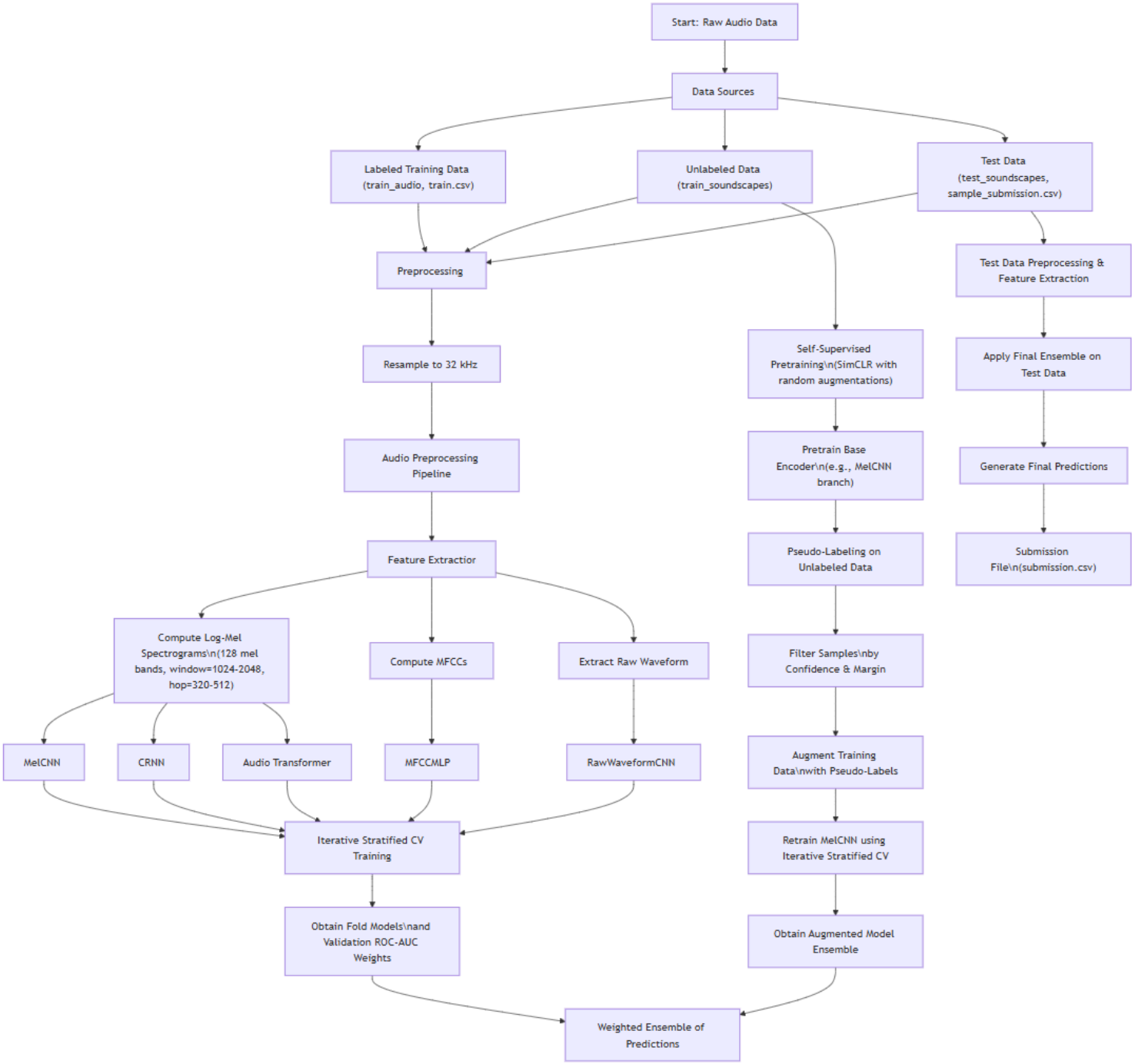
To achieve this goal, we propose to use an ensemble model approach to analyze the animal vocalization data. By using an ensemble method approach we can capture different aspects of the data to increase the accuracy of our final score. All the data sources will go through five different neural networks. These networks will be a CNN on mel spectrogram, CRNN, an audio transformer, MFCC+MLP, and a raw waveform CNN. We will then average the results of all the neural networks together to get the final score. By using the ensemble model approach we hope to get better performance and accuracy then we would get from using just one neural network.

The Data: https://www.kaggle.com/competitions/birdclef-2025/data

For this project, we will be using the dataset that is provided to us by the Kaggle competition organizers. It comprises three different sources of animal vocalization data. The sources are xeno-canto.org, iNaturalist and the Colombian Sound Archive of the Humboldt Institute for Biological Resources Research in Columbia. Xeno-canto.org and iNaturalist are websites where end users can upload their own animal recordings. Their primary focus is on animal identification. Each of their datasets consists of user created recordings of animals that they find around them. The dataset for this project consists of a smaller subset of those datasets. This dataset specifically consists of recordings of various birds, amphibians, mammals and insects all found within the Middle Magdalena Valley of Colombia. The data is split up into four different sections. These sections are called train_audio, test_sondscapes, train_soundscapes and hidden test data. For the train_audio, the files consist of the following information. The first feature is the primary_label which specifies the code for the species. In this case it is eBird code for birds and iNaturalist taxon ID for everything else. In total there are 206 species that can be identified in the training data. There are also 28,564 labeled training examples available to us in this dataset. Then a recording may have a feature called secondary_labels. This is for when there are other species of animals that can be identified in the recordings but they are not the primary focus. They may not all have secondary labels. Then they have a feature called type and in some of the samples it explains what type of call an animal is making. This may describe how the sound is meant to scare off other animals or call fellow animals to a particular place. The next feature is the filename. This is the filename of the sound files. This links the sound files to the spreadsheet so that they have some background context to what is being heard in the audio. The next feature is called collection, and it indicates the source of the data and what the recording's id was in that collection. Some recordings also provide a quality rating if the data comes from the Xeno-canto dataset. If there are no ratings, then the number 0 will be used to indicate that no rating is available. The next feature is the url of where the sound file came from. After that the next feature is the geographical location of the recording. These are given as latitudinal and longitudinal coordinates. The next feature is called scientific_name and this is just the scientific name of the animal that was heard in the sample. Likewise, the next feature is called common_name and this is the common name of the animal in the recording. Then the next feature is called author. If an author of the file can be specified it will be found in this column. If there is no author then the name "Unknown" will be used in place of the actual author's name. The last feature is just the license for the sound file. In total there are 13 features. For the test_soundscapes data, there are approximately 700 1 minute recordings to process. They are all resampled to 32 kHz. Not all species from the train data occur in this data. Along with test_soundscapes there is also train_soundscapes. This data is unlabeled, but it comes from the same location as test soundscapes. They do not overlap with the hidden test data. In this dataset there are 9,726 samples to analyze. Through the combination of datasets between the three different sources and the ensemble approach we will take to analyze the data, the model should be able to achieve a high accuracy when tested with the hidden test data.

Methods: Ensemble Model Architectures for Species Classification:

We propose an ensemble of **multiple neural network architectures**, each tailored to different feature types and aspects of the bioacoustic data. This diversity will help the model generalize from limited samples imageclef.org.



The ensemble will include:

- **CNN on Mel-Spectrograms:** A convolutional neural network that treats the mel-spectrogram as an image. For example, an **EfficientNet** or **ResNet** backbone (pretrained on ImageNet, to leverage transfer learning) can be adapted to spectrogram inputs by using 1-channel inputs. Such CNNs will learn frequency-temporal filters (e.g. to detect bird chirp patterns or frog croaks) and are typically very strong classifiers for audio spectrograms. We will fine-tune these models on the BirdCLEF data; pretrained image models give a head-start on feature extraction even with few labels. In past bird audio competitions, ensembles of CNNs with different backbones have achieved top performance blog.csdn.net, so we will include multiple CNN variants (e.g. EfficientNet-B3, ResNet34, etc.) each trained on spectrograms with slightly different settings (different time crop lengths, frequency ranges, or augmentation settings) to provide complementary predictions.

- **CRNN (Convolutional Recurrent Network):** To capture temporal dynamics beyond the CNN's receptive field, we include a model combining CNN layers (for feature extraction from the spectrogram or MFCC sequence) with a **Recurrent layer (LSTM/GRU)** or **Temporal Attention** mechanism. The CNN learns local spectral features, and the RNN/attention summarizes how calls evolve over time, which is useful for species with distinctive rhythm or periodic calls. This CRNN architecture can operate on mel-spectrogram patches or MFCC sequences. It is optimized for sequences, helping to detect a species call that might be a few seconds long within a longer clip. With limited data, we will constrain model size (e.g. a small GRU) and use dropout to prevent overfitting.

- **Audio Transformer:** Leverage a transformer-based architecture (such as an **Audio Spectrogram Transformer** or a **Perceiver**) that can model long-range context in the audio. Transformers have shown success in audio classification by attending to relevant time-frequency regions. We could use a pretrained AudioSet model (e.g. AST or PANNs CNN14 transformer variant) if allowed, or train a transformer from scratch on our data augmented with unlabeled examples. This model will be adept at capturing complex patterns like overlapping calls from multiple species. It provides a different inductive bias compared to CNNs (global self-attention vs. local convolution), enriching the ensemble. Given the data limits, we might freeze parts of a pre-trained transformer and fine-tune just the later layers to reduce the risk of overfitting.

- **Raw Waveform CNN:** A 1D convolutional network operating on raw wave input (with possible initial SincNet or wavelet-based filters). This model will learn frequency-selective filters automatically. To make it feasible with limited labels, we will initialize it from a self-supervised pre-training on unlabeled audio (detailed in the next section). The architecture could resemble an encoder from a known model (e.g. a smaller WaveNet-like stack or the front-end of wav2vec). This provides the ensemble an *end-to-end* learned feature extractor that might capture subtle cues (like precise pitch or timbre cues) that fixed feature transforms might miss. We will ensure this model remains lightweight (fewer layers) so that it can train with the data available and run efficiently.

- **MFCC+MLP Classifier (Lightweight model):** In addition to deep models, we can include a simple **machine learning model** on MFCC features or other handcrafted features. For instance, a small **multi-layer perceptron** or even a **Gradient Boosted Trees (XGBoost)** model can take as input aggregated MFCC statistics or species-specific acoustic indices. This model is less powerful than the neural nets but can excel when data are very scarce, since MFCCs distill the audio and the model has fewer parameters. It acts as a safety net in the ensemble – because it's less likely to overfit, it may perform more consistently on rare classes. Its predictions can slightly boost ensemble performance especially on species where deep nets struggle due to extremely few examples.

Each model is **optimized for limited labeled data** by leveraging transfer learning or careful regularization. All CNN/transformer models will use pretrained weights (ImageNet or AudioSet) whenever possible to reuse learned audio-relevant features arxiv.org. We will apply strong augmentation (random noise, time-stretch, pitch shift, mixup of background audio) during training to expand the effective dataset size. Additionally, techniques like **mixup** and **balanced sampling** (oversampling rare species) will address class imbalance. The architectures will be tuned on validation data to avoid overfitting – e.g., using early stopping if no improvement. By combining architectures (CNNs, RNNs, transformers, etc.), we ensure the ensemble captures complementary aspects of the data (localized features, temporal dynamics, long-range context, and different feature representations). Prior competitions have shown that an ensemble of diverse models can significantly outperform any single model blog.csdn.net.

Ensemble Method / Technique to Maximize ROC-AUC

Once we have multiple trained models, we need to combine their predictions effectively to maximize the ROC-AUC metric. We will evaluate several ensemble techniques and likely use a **weighted averaging (blending)** approach with optimized weights:

- **Weighted Averaging (Blending):** The simplest and often most robust method is to average the prediction probabilities of each model in the ensemble. Rather than a plain average, we will assign higher weights to models that performed better on validation (higher AUC) and to models that complement each other. We can determine the weights by analyzing out-of-fold prediction performance: for example, solve an optimization (even a simple linear regression or grid search) to maximize AUC on the combined OOF predictions. In one audio competition, a weighted blending based on out-of-fold performance was used to boost ensemble score huggingface.co. We will do similar – using our CV results to find, say, model A should get 0.4, B 0.3, C 0.3 in the average (or even use a geometric mean for stability huggingface.co). This blending will be done at the probability level for each species. Weighted averaging is fast at inference and usually avoids overfitting, making it suitable for the final Kaggle submission.

- **Stacking (Meta-learner):** As an extension, we could train a small meta-model on the validation predictions (stacking) to combine models in a non-linear way. For instance, a logistic regression could take each model's output score for a clip and learn the optimal combination to predict the true label. This can sometimes edge out a simple blend in ROC-AUC. However, given the limited data (especially for rare classes), a complex

meta-learner might overfit the validation set. We will try stacking in offline experiments, but it must be used with caution. If stacking, we'd use a simple regularized model (like logistic regression or linear blend) and possibly cross-validate the stacking itself. In a code competition, implementing stacking means our submission must carry an extra model training step for the meta-learner, which is additional complexity.

- **Majority Voting / Rank Averaging:** Since ROC-AUC is threshold-independent, we primarily care about ranking of positives vs negatives. We might also consider rank averaging (averaging the rank of each prediction across models) or a voting scheme for presence/absence. These are less likely to be optimal than a learned weighting, but are fallback options if probability calibration is an issue. Our plan is to trust the calibrated probabilities of each model and blend them.

Ultimately, we expect a **stacked generalization via weighted averaging** to be our ensemble solution: for each test audio clip, each model outputs a probability for each species, and we take a weighted sum of these probabilities as the final score. This ensemble prediction will then be used to calculate the ROC-AUC. By combining multiple models, each species' signal has multiple chances to be detected, improving overall AUC. We will also validate that the ensemble's AUC on our cross-val sets exceeds any single model's AUC (to ensure ensembling is actually helping, which it typically does when models are diverse).

Audio Feature Extraction Techniques

To robustly capture bird, amphibian, mammal, and insect sounds, we will combine multiple audio feature representations:

- **Mel-Spectrograms:** Convert audio into mel-frequency spectrogram images (time-frequency representations). These preserve rich frequency patterns (e.g. bird chirps, insect stridulations) and are well-suited for CNN-based classification. Mel-spectrogram features have proven highly effective for deep learning in audio tasks ieeexplore.ieee.org. We will use log-mel spectrograms with appropriate parameters (e.g. 32 kHz sample rate, 128 mel bands, window ~1024-2048, hop ~320-512) to capture relevant frequencies of target species. We may also experiment with Per-Channel Energy Normalization (PCEN) or different time-frequency resolutions to optimize for various call types (short chirps vs. long calls).

- **MFCCs (Mel-Frequency Cepstral Coefficients):** Extract MFCC features as a compact summary of the audio's spectral shape. MFCCs capture perceptually relevant timbre information and are relatively robust to noise and channel distortions ideas2it.com, which is useful in diverse outdoor recordings. We can compute 20-40 MFCC coefficients over time frames and use them in two ways: (1) feed sequences of MFCCs into a recurrent or convolutional model, or (2) aggregate statistics of MFCCs as features for a lightweight classifier. Including MFCC-based models adds diversity to the ensemble, as they may capture features (e.g. broad spectral envelope) that complement spectrogram-based models.

- **Raw Waveform Processing:** Incorporate models that learn directly from raw audio waveforms. End-to-end waveform models can learn custom filters (e.g. via an initial convolutional layer or SincNet/learnable filterbank) to extract features optimal for the data. This avoids hand-crafted feature biases and could capture phase or temporal details lost in spectrograms. However, training from raw audio typically requires more data to outperform spectrograms
  reddit.com. To mitigate this with limited labeled data, we will **pretrain** the raw-wave model on unlabeled audio (self-supervised, see below) so it can learn useful filters. By including a raw waveform branch (e.g. a 1D CNN or WaveNet-style model), the ensemble gains another perspective that might detect subtle wave patterns or temporal transients that spectrograms smooth out. In practice, spectrogram-based approaches are expected to dominate with small data reddit.com, but a pre-trained waveform model could contribute unique signals for certain species.

Implementation: We will use **Librosa** (or similar) in the Kaggle environment to compute spectrograms and MFCCs on the fly or offline. To stay within runtime limits, we can precompute and store these features as Kaggle datasets. We'll apply standard augmentations during feature extraction (random time shifts, background noise, etc.) to increase effective training data. All feature pipelines will be encapsulated in the code and use only the provided audio data (complying with competition rules).

Self-Supervised and Semi-Supervised Learning Integration

To fully exploit the unlabeled audio provided (as encouraged by the competition) imageclef.org, our strategy integrates both self-supervised pre-training and semi-supervised training:

- **Self-Supervised Pre-training:** We will train a neural **audio encoder** on the unlabeled audio recordings using self-supervised learning (SSL). The encoder could be a CNN or transformer that processes either raw wave or spectrogram inputs. We will adopt contrastive learning approaches such as **SimCLR or BYOL** on audio snippets: the model will be trained to produce similar embeddings for two augmented versions of the same audio clip and dissimilar embeddings for different clips. This way, the model learns to recognize inherent acoustic patterns without labels. In bioacoustics, contrastive SSL has been shown to yield representations that improve downstream classification, especially when labeled data are scarce arxiv.org. For example, we can take 5-second crops from the unlabeled soundscapes, apply augmentations (random time shift, filter, noise), and train the network to maximize agreement between embeddings of the same clip (positive pair) and minimize it for others (negative pairs) arxiv.org. This training will teach the model to encode general bird/insect/mammal sound characteristics (such as tonal vs. broadband calls, ambient noise patterns, etc.). We will use the learned encoder weights to initialize our spectrogram CNNs and/or the raw waveform model. This initialization gives a head start, requiring the model to only fine-tune to the specific species classification task rather than learning from scratch.

- **Unsupervised Pretrained Models:** In addition to our own SSL, if the competition permits, we will leverage existing pre-trained models as a form of self-supervised feature extraction. For instance, **BirdNET**

embeddings (a model trained on thousands of bird audio hours) are known to provide powerful features for bird calls arxiv.org. We could run BirdNET (or a Google *Perch* model) on our audio to get embedding vectors, and feed those into a new classifier trained on the BirdCLEF labels. This is effectively transfer learning from a massive external dataset, which aligns with self-supervision goals. **Note:** We will only do this if allowed by competition rules (using external models/data); otherwise, our own SSL pretraining on provided data will be the focus.

- **Semi-Supervised Learning (using unlabeled data):** Beyond pretraining, we will directly incorporate unlabeled data into the training loop via **pseudo-labeling** and **consistency training**:

    - *Pseudo-Labeling:* We will train an initial version of our model (or ensemble) on the labeled data, then use it to predict labels on the unlabeled audio. The highest-confidence predictions for each species will be treated as "pseudo-labels." We then combine these pseudo-labeled samples with the real labeled data to retrain/refine the models. This effectively increases the training set size. We will do this iteratively: each round, add new confident predictions to the training set and retrain, which can gradually improve the model's accuracy as it learns from more data. This technique has improved performance in past audio competitions; for example, in an audio tagging challenge, soft pseudo-labeling boosted validation scores significantly (e.g. CV score from 0.849 to 0.870 in one case) huggingface.co. We will be cautious to avoid reinforcing errors – only very confident predictions (or using ensemble agreement) will be added, and we'll monitor that the public LB doesn't degrade (to avoid overfitting to noise huggingface.co).

    - *Consistency / Teacher-Student Training:* We will also explore a mean-teacher approach or FixMatch-like strategy, where the model learns to be consistent on unlabeled data under perturbations. For example, for an unlabeled audio clip, we obtain a "teacher" prediction using the current model (with no augmentation), then train the model ("student") to predict the same output when the clip is augmented (e.g. adding noise or time-shift). This encourages the model to develop stable predictions and can leverage unlabeled data without explicitly assigning hard labels. Such semi-supervised consistency training can refine decision boundaries using the distribution of unlabeled examples.

By integrating these methods, the limited labeled data can be augmented with **information gleaned from the abundance of unlabeled recordings**, improving the model's ability to recognize rare species. Our final training pipeline will likely intermix these approaches – e.g. first self-supervised pretrain on all unlabeled, then supervised train on labeled, then a round of pseudo-label fine-tuning. This addresses the core challenge of BirdCLEF+ 2025: achieving high accuracy with very few labeled examples by *teaching the model from the data itself* imageclef.org.

Cross-Validation Strategy for Robust Generalization
A careful cross-validation (CV) scheme is essential to reliably gauge performance and to maximize the use of limited data. We will design a CV that is stratified and group-aware, while mindful of Kaggle's computational constraints:

- **Stratified K-Folds:** We plan to use a 5-fold stratified cross-validation by species. Given the class imbalance (rare species with few recordings), stratification ensures each fold's training set has at least some examples of each species (where possible). If the task is multi-label (multiple species in one clip), we will use iterative stratification to maintain label distribution across folds. In BirdCLEF 2023, it was noted that a naive fold split could omit certain classes in a fold; a solution was to drop that fold from validation blog.csdn.net. We will verify that each fold covers all or most species – if one fold is missing a rare species entirely, we might adjust (e.g. use 4-fold CV instead, as one team did blog.csdn.net). Our splits will be fixed and used consistently for model development and blending.

- **Group-wise Splitting:** The BirdCLEF data includes soundscape recordings from various locations/times. To avoid "evaluation on very similar data" (which could overestimate performance), we'll implement a **group split** by recording or location. For example, if multiple clips come from the same continuous recording or the same site, we assign them to the same fold as a group. This way, no fold has nearly identical background or noise conditions as another, better simulating the generalization to new environments. Grouped CV prevents leakage of ambient background cues between train and val. We will check metadata (e.g. recording ID or sensor location) to define these groups.

- **Temporal/Habitat Considerations:** If the dataset has seasonal or habitat variations, we ensure those are represented in each fold to avoid bias. In practice, our stratified group k-fold will attempt to balance by species while grouping by recording source.

- **Use of Cross-Val in Model Training:** We will train a separate instance of each model for each fold (when computationally feasible). That yields five models per architecture, which we later ensemble. This approach uses all the data for training (each sample is in 4 of 5 training sets) and also provides out-of-fold predictions for ensemble weighting. However, training 5 folds for all architectures can be time-consuming. To remain within Kaggle's time limits, we might do full CV training offline and only use the pre-trained fold models in the submission (loading their weights). If required to train on Kaggle, we could reduce to e.g. 3-fold CV or train a subset of models per fold due to the 9-hour execution cap.

- **Efficiency Considerations:** We will use CV primarily to tune hyperparameters and to generate ensemble candidates. If constrained by Kaggle runtime, an alternative is **cross-validation by model type** – e.g. do 5-fold for the primary CNN model (to get reliable performance estimate and variance), but train some other models on the full training set if they're low-capacity, to save time. We will ensure that our final model selection is informed by CV (choosing architectures that perform best on average across folds). The validation scheme also serves to compute blending weights (using out-of-fold predictions from each model on its respective val fold to optimize ensemble weighting, described next).

By employing stratified group k-fold, we maximize generalization and resilience to overfitting. The ensemble will be built on models that have all been validated on unseen folds, giving us confidence in their generalization. This CV strategy balances rigor with practicality under Kaggle's constraints by possibly limiting the number of folds or parallelizing fold training within the allowed timeframe.

Training and Implementation Details (Kaggle Compliance)

To ensure our approach is executable under Kaggle's code competition constraints (compute limits and code requirements), we will carefully manage the pipeline:

- **Compute Resources:** We anticipate using the provided GPU acceleration (likely one high-end GPU, limited runtime ~8-9 hours). Training all models from scratch within this limit is challenging. To maximize performance, we will do heavy training and tuning in offline sessions; then we'll **save the trained weights** for each model (possibly in a Kaggle dataset). The submission code will primarily handle loading models and running inference on the test set (which is feasible within time). For example, in BirdCLEF 2023 a 4th-place solution loaded multiple pre-trained fold models as Torch Script files and ensemble them in the Kaggle runtime blog.csdn.net. We will adopt a similar approach: save each neural network as a serialized artifact (PyTorch .pt or ONNX), load them in the Kaggle notebook, and perform the forward passes on test audio. This complies with the *"code competition"* rules as training code is included (or referenced) and weights are provided as allowed data.

- **Code Organization:** We will structure the code to have clear sections for feature extraction, model definitions, training (if any done live), and inference/ensemble. We must adhere to competition rules such as no internet access and using only provided or competition-allowed data. Any external libraries or pre-trained models we use will either be from Kaggle's standard environment or added to the environment beforehand (e.g. as data files). We'll rely on common libraries like **PyTorch, TensorFlow, librosa, scikit-learn** which are available. The code will be thoroughly tested on a subset of data to ensure it runs within memory and time limits.

- **Speed Optimizations:** To maximize speed, we will use mixed-precision (FP16) inference, and vectorized batch processing of audio data. If test audio are long soundscapes, we'll split them into segments (e.g. 5-10s) and run the models on each segment with overlapping windows, then aggregate the segment predictions (taking max or mean) to get clip-level results. This sliding-window inference will be implemented efficiently to avoid redundant computation. We will also cache intermediate features: for instance, compute mel-spectrograms once and reuse them for all spectrogram-based models to avoid duplicate work. If multiple models share a backbone architecture, we might load the backbone once and reuse it if possible, or use model quantization to reduce size. Some top teams have used model quantization and optimized inference engines (like ONNX Runtime or OpenVINO) to speed up ensembles blog.csdn.net. We can consider converting our PyTorch models to ONNX and using multi-threaded CPU for the smaller models, keeping GPU for the heavy

CNNs, to parallelize inference across resources.

- **Memory Management:** Loading 5+ models can consume a lot of memory. We will choose efficient architectures (e.g. EfficientNet-B0/B3 rather than very large nets) and possibly load one model at a time if needed (though that slows inference). Another trick is to load models in half precision to halve memory usage. Using Touchscript/ONNX will also eliminate the overhead of Python model instantiation. We'll test ensemble inference memory usage to ensure it fits in Kaggle's GPU (typically ~16GB).

- **Compliance:** All datasets (including any pretrained weight files or unlabeled data) will be attached via the Kaggle datasets interface. We won't use any disallowed external data – only the provided BirdCLEF data and possibly pre-trained models from recognized sources if the rules permit (and if not, we stick to self-supervised on provided data). Our methodology of using unlabeled data is explicitly aligned with the competition goals imageclef.org. We will make sure to respect any package installation limits (avoid heavy non-standard packages). The final code will be clear, with commented sections explaining each step (which is useful for the required **working notes** paper as well).

- **Cross-Validation in Code:** If we cannot train all folds in the submission, we will still include the code for training one model (to show methodology) and simply load the pre-trained fold models for actual predictions. This demonstrates compliance (the code contains the training procedure) while leveraging pre-computed results for performance. Kaggle typically allows such practice as long as everything needed to reproduce is provided.

- **Maximizing Performance within Limits:** We will use as much of the allowed time as possible to run inference with TTA (test-time augmentation). For example, we can augment test audio (slight time shifts or noise) and average predictions to smooth out randomness. We will ensure the ensemble's inference is still within time (e.g. if 5 models each need to process ~1 hour of audio, we must budget that carefully). By profiling our code on a subset, we'll optimize any slow points (I/O bottlenecks reading audio, etc.). If needed, we downsample audio (e.g. to 16 kHz) for models that don't need full bandwidth, to reduce computation.

In summary, this strategy brings together **state-of-the-art audio processing techniques, an ensemble of complementary models, and advanced semi-supervised learning** to tackle BirdCLEF+ 2025. By extracting multiple feature types and leveraging unlabeled data, our models can recognize species with very few examples imageclef.org. A carefully crafted cross-validation and ensembling approach will ensure the solution is robust and achieves high ROC-AUC. All components are designed with Kaggle's practical constraints in mind, so the final solution is not only high-performing but also reproducible and efficient within the competition environment. With this plan, we aim to maximize performance while fully complying with competition requirements, paving the way for a top-tier BirdCLEF+ 2025 submission.

Sources:

1. BirdCLEF+ 2025 Competition Description – Limited labels and unlabeled data emphasis
   imageclef.org

2. Effectiveness of mel-spectrogram features with CNN models
   ieeexplore.ieee.org

3. Advantages of MFCCs (noise and channel robustness)
   ideas2it.com

4. Discussion on raw waveform vs. spectrogram for deep learning
   reddit.com

5. Success of diverse CNN ensemble in BirdCLEF (2023)
   blog.csdn.net

6. BirdCLEF 2023 4th place – cross-validation and fold coverage detail
   blog.csdn.net

7. Self-supervised contrastive learning benefits for audio representations
   arxiv.org

8. Pre-trained bird audio embeddings aiding few-shot classification
   arxiv.org

9. Pseudo-labeling improving audio classification performance
   huggingface.co

10. Weighted ensemble blending using OOF optimization
    huggingface.co

11. Example of loading multiple model checkpoints for ensemble (Kaggle code)
    blog.csdn.net