Predicting Spotify Skips from Song Intros

Comparing Handcrafted Features, CNNs, Audio Transformers, and Context Fusion

Pedro Henrique Goncalves de Paiva

April 24th, 2026

Abstract

This project asks two related questions. First, can the first 10 seconds of a song predict whether I will skip it? Second, if audio alone is limited, how much does performance improve when I add listening context? I use my personal Spotify streaming history, containing 92,445 play events from 2015 to 2026, to construct track-level skip labels. Because Spotify does not provide raw audio through this workflow, I match Spotify tracks to Deezer preview clips and use the first 10 seconds of each matched preview as the audio input. The final matched datasets contain 2,892 loose-label clips and 1,169 strict-label clips.

I compare handcrafted audio features, frozen Audio Spectrogram Transformer embeddings, a CNN trained on log-mel spectrograms, a fine-tuned AST model, context-only baselines, and multimodal fusion models. The audio-only models answer the original question about whether song intros contain predictive signal. The context-only and fusion models act as diagnostic controls: they test whether skip prediction improves because of audio, because of listening context, or because the two sources provide complementary information.

The main conclusion is therefore not that 10-second audio intros solve skip prediction. A more accurate conclusion is that audio contains limited but meaningful signal, while the context and fusion comparisons reveal how much of my skip behavior depends on information outside the waveform. After the final run, I report the best audio-only, context-only, and fusion results separately rather than treating them as the same task.

Contents

1 Executive Summary

The central question of this project has two layers. The main audio-only question is: given only the first 10 seconds of a song, can a machine learning model predict whether I will skip it? The diagnostic extension is: if audio alone is not enough, what changes when I add listening context? This distinction matters because a model using only sound is solving a different problem from a model using sound plus information about my listening habits. The project therefore first tests whether song intros contain predictive signal, then uses context fusion to diagnose what audio-only models are missing.

The full pipeline is shown in fig. 1. First, I use Spotify Extended Streaming History to construct labels. At the event level, a play below 30 seconds is treated as a skip, a play above 60 seconds is treated as a keep, and the ambiguous 30–60 second range is dropped. At the track level, I aggregate repeated plays into a skip rate. I use two labeling schemes: a loose label, where skip rate ≥ 0.50 becomes Skip, and a strict label, where skip rate ≥ 0.75 becomes Skip and skip rate ≤ 0.25 becomes Keep. The strict label drops the ambiguous middle and becomes the main dataset for the final run.

The main contribution is the model comparison, but I separate the comparison into audio-only models and context-augmented models. The audio-only models answer the original question about the first 10 seconds of sound. The context-augmented models are a diagnostic extension that tests whether skip behavior depends on information outside the waveform. I compare:

1. Handcrafted audio features, such as MFCCs, RMS energy, and zero-crossing rate, passed to shallow models.
2. Frozen AST embeddings, where a pretrained Audio Spectrogram Transformer produces a 768-dimensional representation and shallow models classify it.
3. A scratch CNN trained directly on log-mel spectrograms.
4. A CNN with full augmentation, including Gaussian noise, time shift, and SpecAugment-style masking.
5. A fine-tuned AST model, where the last four encoder blocks and a new classifier head are trainable.
6. Context-only and fusion models, which compare listening context alone against audio plus context.

The results are mixed, which is part of the finding. The strongest strict-label audio-only cross-validation result is Frozen AST embeddings + KNN, with 61.8% ± 2.4% accuracy, 57.7% ± 2.2% balanced accuracy, and 0.597 ± 0.028 AUC. This shows limited audio-only signal, but the interpretation depends on balanced accuracy, AUC, and the dummy majority baseline, not raw accuracy alone. The scratch CNN is competitive at 60.1% ± 1.9% without augmentation, while the augmented CNN reaches 58.0% ± 3.3%. Fine-tuned AST reaches 56.0% on a fixed validation pilot, but I do not treat it as directly comparable to 5-fold CV because it was trained using a computationally cheaper fixed split. The strongest overall cross-validation result comes from context-only Logistic Regression, which reaches 75.1% ± 0.8% accuracy, 74.6% ± 0.6% balanced

accuracy, and 0.817 ± 0.014 AUC. The best fusion model reaches 73.4% ± 2.5% accuracy, 73.1% ± 2.3% balanced accuracy, and 0.790 ± 0.026 AUC. This suggests that context explains most of the gain, and that the current audio representation does not clearly add complementary signal beyond context.

The project therefore does not support the simple claim that "deep audio models solve skip prediction." A better conclusion is that audio contains weak and unstable signal, while listening context explains much more of the task in this dataset. In other words, the first 10 seconds of a song matter, but my skip decision is also shaped by when I listen, how familiar the artist is, whether the song appears in a listening habit, and other variables outside the waveform.

Family	Representation	Model	CV Acc	CV Bal Acc	CV F1	CV AUC
Baseline	None	Dummy majority	61.1% ± 0.2%	50.0% ± 0.0%	0.76 ± 0.00	0.500 ± 0.000
Audio-only	Simple 15-dim features	KNN, k=11	56.7% ± 2.8%	52.7% ± 2.3%	0.66 ± 0.03	0.532 ± 0.033
Audio-only	Frozen AST embeddings	KNN, k=11	61.8% ± 2.4%	57.7% ± 2.2%	0.71 ± 0.02	0.597 ± 0.028
Audio-only	Frozen AST embeddings	SVM, RBF	60.1% ± 2.8%	58.6% ± 2.0%	0.67 ± 0.04	0.614 ± 0.027
Audio-only	Mel spectrogram	CNN, no augmentation	60.1% ± 1.9%	56.2% ± 3.4%	0.69 ± 0.05	0.596 ± 0.028
Audio-only	Mel spectrogram	CNN, full augmentation	58.0% ± 3.3%	56.2% ± 2.3%	0.64 ± 0.07	0.601 ± 0.020
Audio-only pilot	Mel spectrogram	Fine-tuned AST	56.0% ± 0.0%	56.4% ± 0.0%	0.60 ± 0.00	0.610 ± 0.000
Context-only	Listening context	Logistic Regression	75.1% ± 0.8%	74.6% ± 0.6%	0.79 ± 0.01	0.817 ± 0.014
Fusion	AST embeddings	AST + context (Logistic Regression)	66.4% ± 3.0%	65.0% ± 2.7%	0.72 ± 0.04	0.706 ± 0.024
Fusion	CNN penultimate features	CNN features + context (Logistic Regression)	73.4% ± 2.5%	73.1% ± 2.3%	0.77 ± 0.03	0.790 ± 0.026

Table 1: Main strict-label cross-validation comparison from the final notebook run. Audio-only rows answer the original 10-second audio question. Context-only and fusion rows test whether listening context explains signal that audio alone misses.

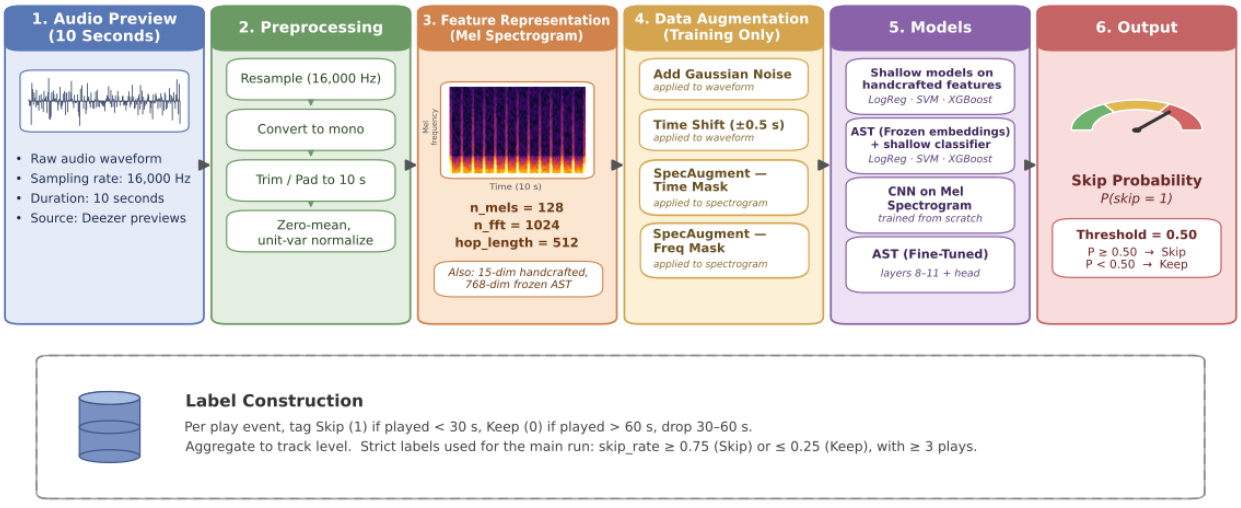

Figure 1: Overall project pipeline. The pipeline starts with a 10-second Deezer audio preview, preprocesses it into a log-mel spectrogram, optionally applies training-only augmentation, and evaluates several audio-only models plus context-fusion extensions. Spotify listening history is used separately to construct the Skip/Keep labels.

2 Introduction

Skipping a song looks like a clean binary outcome: I either skip or I do not skip. But behaviorally, it is messier than that. I may skip the same song in one setting and listen to it in another. A skip may reflect the song's intro, but it may also reflect mood, time of day, shuffle behavior, device, or artist familiarity. That makes this a useful machine learning problem because it is personal, noisy, and not obviously solvable by a single model.

In my previous CS156 assignment, I modeled skip prediction using Spotify-derived metadata and pretrained embeddings. That project worked as a pipeline, but it had a limitation: the model was not listening to the song. It mostly learned from compact variables and embeddings. For the final project, I move closer to the raw data by using audio previews. I separate the project into two related questions:

> *Audio-only question: Can the first 10 seconds of a song predict whether I will usually skip it?*
>
> *Diagnostic extension: If audio-only models are limited, how much does prediction improve when I add listening context?*

My thesis is deliberately cautious:

> *Richer audio representations should capture more of the song intro than handcrafted summaries, but audio alone may not generalize well because skipping is partly a context-dependent personal behavior. If context-only or context-fusion models perform much better than audio-only models, that should be interpreted as evidence that the original audio-only task is missing important behavioral information.*

This thesis leads to two predictions. First, models that preserve more audio structure, such as CNNs and AST, should often beat very compressed audio features. Second, if audio-only models remain weak while context-only or context-fusion models improve, that failure is meaningful. It would suggest that the missing signal lives outside the first 10 seconds of audio.

3 Data

3.1 Source Data

The label source is my Spotify Extended Streaming History, which contains 92,445 play events from 2015 to 2026 across 6 JSON files. Each event includes information such as track name, artist name, timestamp, platform, milliseconds played, and reason for ending the stream. This satisfies the personal-data requirement of the assignment because it comes from my own digital archive. At the same time, the final paper does not expose private content. It uses aggregate labels and song-level information.

The audio source is Deezer preview clips. Spotify does not provide raw song audio through this workflow, so I searched Deezer using normalized track and artist names, downloaded matched preview clips, and used only the first 10 seconds. This creates one important limitation: the label comes from Spotify behavior, but the audio may come from a Deezer preview version. To reduce mismatch, the code filters out obvious live, remix, acoustic, karaoke, cover, extended, demo, and unplugged versions when they do not match the Spotify title.

3.2 Label Construction

The label construction is shown visually in fig. 2.

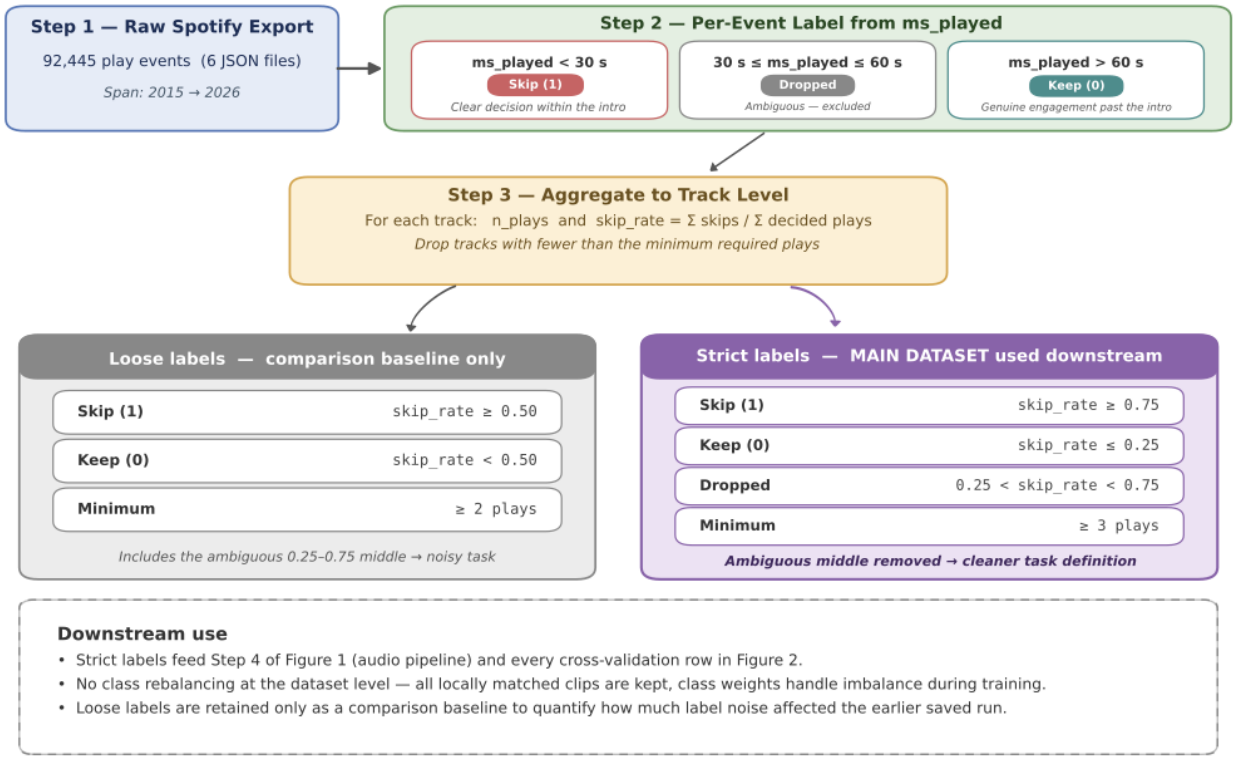

Figure 2: Label construction pipeline. Spotify play events are converted into event-level Skip/Keep labels using playback duration, then aggregated into track-level skip rates. The strict label removes the ambiguous middle and becomes the main downstream dataset.

At the event level, I use playback duration to infer whether a play was a skip or a keep. Let t_i be the number of seconds played in event i. The event label is:

$$y_i = \begin{cases} 1, & \text{if } t_i < 30 \text{ seconds} \\ 0, & \text{if } t_i > 60 \text{ seconds} \\ \text{dropped}, & \text{if } 30 \leq t_i \leq 60 \end{cases}$$

Here, 1 means Skip and 0 means Keep. The 30–60 second range is dropped because it is

ambiguous. A 45-second play could mean I listened briefly, got interrupted, scrubbed the track, or skipped late. Keeping those events would make the label less clean.

At the track level, I aggregate all decided events. For a track j with n_j decided plays, the skip rate is:

$$\text{skip\_rate}_j = \frac{1}{n_j} \sum_{i=1}^{n_j} y_{ij}.$$

A concrete example makes the formula clearer. Suppose I played one song 5 times, and the event labels were:

$$[1, 1, 0, 1, 0].$$

That means I skipped it 3 times and kept it 2 times. The skip rate is:

$$\text{skip\_rate} = \frac{1+1+0+1+0}{5} = \frac{3}{5} = 0.60.$$

Under the loose rule, this track would be labeled Skip because $0.60 \geq 0.50$. Under the strict rule, it would be dropped because $0.25 < 0.60 < 0.75$. This is the main reason strict labels are more honest: they avoid pretending that ambiguous songs are clean examples.

3.3 Loose and Strict Labels

I use two track-level label definitions:

$$\hat{y}_j^{\text{loose}} = \begin{cases} 1, & \text{if skip\_rate}_j \geq 0.50 \\ 0, & \text{if skip\_rate}_j < 0.50 \end{cases}$$

and

$$\hat{y}_j^{\text{strict}} = \begin{cases} 1, & \text{if skip\_rate}_j \geq 0.75 \\ 0, & \text{if skip\_rate}_j \leq 0.25 \\ \text{dropped}, & \text{if } 0.25 < \text{skip\_rate}_j < 0.75. \end{cases}$$

The final matched sample contains 2,892 loose-label tracks and 1,169 strict-label tracks. From here onward, the strict-label set is the main dataset, while the loose-label set is used as a comparison baseline.

Dataset	Candidate pool	Matched	Match rate	Keep	Skip
Loose labels	6,112	2,892	47.3%	1,176	1,716
Strict labels	2,081	1,169	56.2%	455	714

Table 2: Audio dataset summary after Deezer matching. Match rate is computed from the candidate pool after label construction and version filtering.

4 Exploratory Data Analysis

The first EDA question is whether the labels make behavioral sense. Figure 3 shows three views: plays per track, skip-rate distribution, and event durations. The plays-per-track plot is heavily right-skewed, meaning many tracks have only a few plays while a smaller number are repeated often. This matters because a skip rate based on 3 plays is noisier than a skip rate based on 30 plays.

The skip-rate distribution shows many tracks in the middle. This supports the strict-label strategy. If many tracks have skip rates near 0.50, then the model is being asked to classify songs that even my own behavior does not classify consistently. The event-duration view also supports the 30-second and 60-second cutoffs because very short plays are behaviorally different from longer listens.

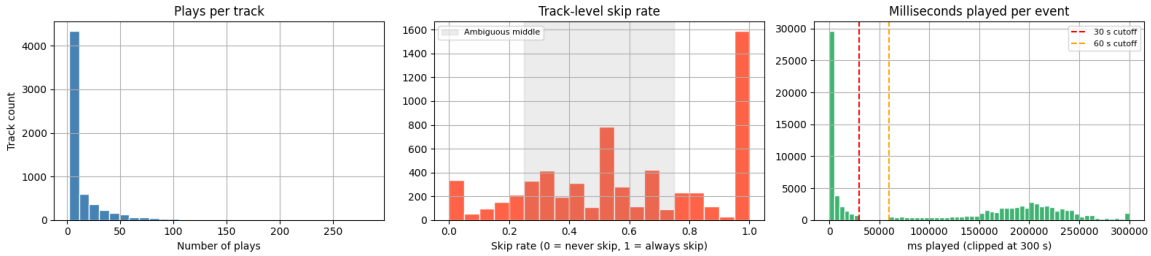

Figure 3: Label validation views. This notebook output checks whether the label definition is reasonable. The left plot shows that plays per track are strongly skewed. The middle plot shows track skip rates and the ambiguous middle region removed by strict labels. The right plot shows event durations and the 30-second and 60-second cutoffs.

The second EDA question is whether skipped and kept songs visually differ as spectrograms. Figure 4 compares one skipped song and one kept song using waveforms and mel spectrograms. The differences are only suggestive, not definitive. The skipped example appears more uniform, while the kept example shows more dynamic energy and frequency variation. This visual inspection motivates the CNN, but it also warns against overclaiming. A spectrogram may show structure, but it does not show mood or context.

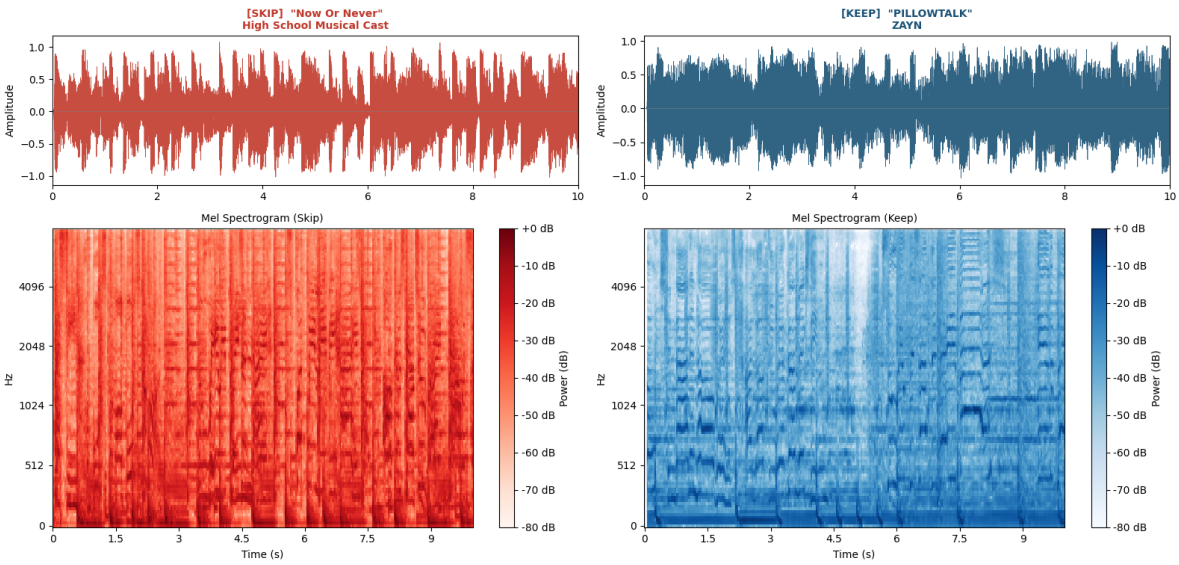

Figure 4: Skip versus keep audio examples. The top panels show waveforms, and the bottom panels show log-mel spectrograms. The goal is not to prove the model visually, but to make the input representation concrete. Brighter areas represent more energy at a particular frequency and time.

5 Feature Representations

5.1 Handcrafted Audio Features

The first representation uses 15 handcrafted audio features:

- 13 MFCC means,
- mean RMS energy,
- mean zero-crossing rate.

This gives each song a vector $x \in \mathbb{R}^{15}$. Each coordinate summarizes one aspect of the 10-second clip. For example, MFCCs summarize the broad spectral shape of the audio, which is why they are commonly used as compact representations of sound (Davis and Mermelstein, 1980). RMS summarizes loudness, and zero-crossing rate loosely tracks how quickly the waveform changes sign, which can be related to noisiness or high-frequency content. I compute these descriptors using standard Python audio-analysis tools from the `librosa` ecosystem (McFee et al., 2015).

This representation is simple and intentionally lossy. It cannot preserve the whole time-frequency structure of a song intro. But it is useful as a baseline because it tests how far we can get using compact summary features.

5.2 Frozen AST Embeddings

The second representation uses a pretrained Audio Spectrogram Transformer. AST treats audio spectrograms as sequences of patches, adapting the transformer idea to audio classification (Gong, Chung, and Glass, 2021). I pass each 10-second clip through the frozen AST encoder and extract a 768-dimensional embedding. This produces:

$$x_j^{\text{AST}} \in \mathbb{R}^{768}.$$

The important idea is that this embedding is not trained from scratch on my small dataset. It is a transfer-learning representation from AudioSet pretraining. In this project, the frozen AST encoder plays the same role that a pretrained image model might play for image classification: it converts raw audio into a richer learned feature vector, then a shallow classifier makes the final skip/keep decision. This follows the general transfer-learning logic of reusing a representation learned from a larger source task and adapting it to a smaller target task (Pan and Yang, 2010).

5.3 Log-Mel Spectrograms

The CNN and AST models use log-mel spectrograms. A mel spectrogram converts audio into a time-frequency image, which makes it possible to apply image-like models such as CNNs to audio. I generate these spectrograms using standard `librosa`-style audio processing tools (McFee et al., 2015). In this notebook, the input shape is:

$$1 \times 128 \times 313.$$

The first dimension is the channel count. The second is the 128 mel-frequency bands. The third is the number of time frames for the 10-second clip after the chosen hop length. The model therefore sees an image-like representation where the vertical axis corresponds to frequency and the horizontal axis corresponds to time.

6 Data Augmentation

The augmentation pipeline is shown in fig. 5. The goal of augmentation is to make the CNN less sensitive to tiny changes in the preview audio while preserving the broad identity of the song. The time and frequency masking steps follow the intuition of SpecAugment, where parts of the spectrogram are masked during training so the model cannot rely too heavily on one exact local pattern (Park et al., 2019). The training branch applies three kinds of augmentation:

1. Gaussian noise on the waveform,
2. time shift on the waveform,
3. SpecAugment time and frequency masks on the spectrogram.

The validation and test branch does not apply augmentation. This is critical. If augmented examples were used in validation or test, then the metric would no longer measure performance on clean, realistic song previews. This keeps augmentation as a training regularization method, not as a way to make evaluation easier.

The Gaussian noise step can be written as:

$$\tilde{x}(t) = x(t) + \epsilon(t), \quad \epsilon(t) \sim \mathcal{N}(0, \sigma^2).$$

In the notebook, σ is proportional to the standard deviation of the clip, for example $\sigma = 0.01 \cdot \text{std}(x)$. If a clip has $\text{std}(x) = 0.20$, then:

$$\sigma = 0.01 \cdot 0.20 = 0.002.$$

A waveform value of $x(t) = 0.15$ might become $\tilde{x}(t) = 0.15 + 0.001 = 0.151$ after a sampled noise value. This is small enough not to change the song identity, but large enough to prevent the CNN from memorizing exact waveform artifacts.

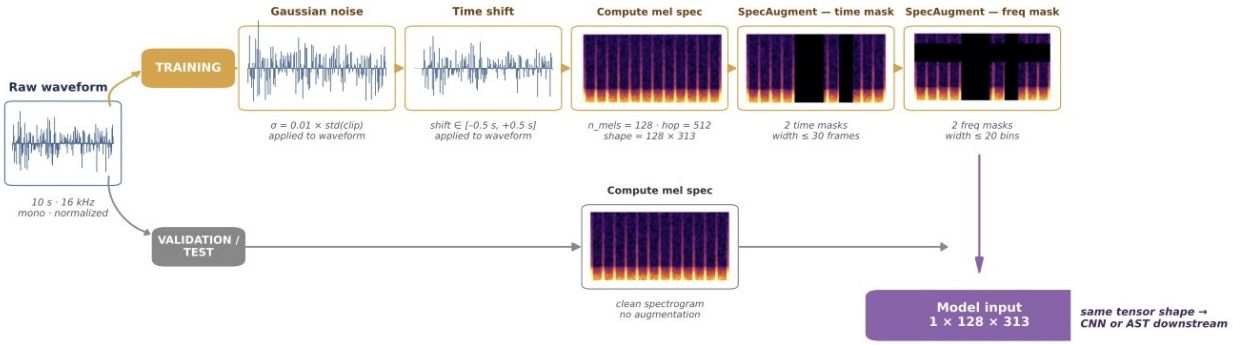

Figure 5: Audio augmentation pipeline. The training branch applies Gaussian noise, time shifting, mel-spectrogram computation, time masking, and frequency masking. The validation and test branch computes a clean spectrogram with no augmentation. Both branches produce the same tensor shape, $1 \times 128 \times 313$, so the same CNN or AST model can process either version.

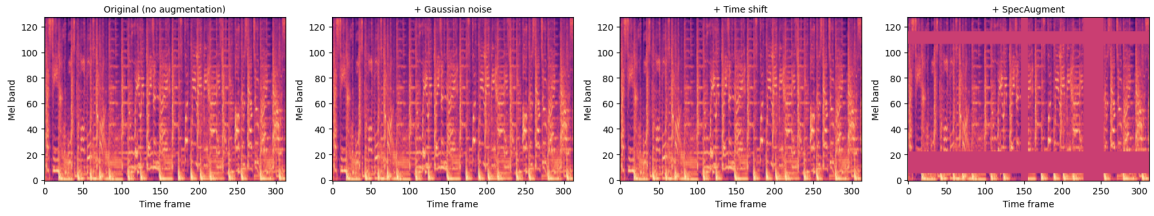

Figure 6: Concrete augmentation examples from the notebook. This figure shows the same song under several augmentation settings. It verifies that the augmentation code changes the input while preserving the broad structure of the clip.

7 Experimental Design

7.1 Train/Holdout Split and Cross-Validation

I use a stratified split so the train and holdout sets preserve the Skip/Keep class balance. The strict set is the main modeling dataset. In the final run, the strict training split contains 935 tracks and the strict holdout split contains 234 tracks. Model selection is based primarily on 5-fold cross-validation within the training split. This is the fairest comparison because every model is evaluated repeatedly on held-out validation folds.

The final split contains 935 training tracks and 234 strict holdout tracks. The full holdout evaluation is implemented in the notebook, but because the final run is computationally expensive, I treat the cross-validation tables as the main model-comparison evidence in this written version. I also include a smaller cached holdout diagnostic for the audio-only finalists. This diagnostic is useful for checking failure modes, but it is not the main model-ranking evidence.

Within the training split, I use stratified cross-validation for model comparison. This means each fold contains roughly the same proportion of Skip and Keep labels. For each fold, the model trains on four folds and validates on the remaining fold. For the shallow baselines, I use standard `scikit-learn` implementations so that the classical models are reproducible and directly comparable across folds (Pedregosa et al., 2011). The reported CV accuracy is:

$$\bar{a} = \frac{1}{K} \sum_{k=1}^{K} a_k,$$

where $K = 5$ and a_k is the validation accuracy on fold k. The standard deviation is:

$$s_a = \sqrt{\frac{1}{K-1} \sum_{k=1}^{K} (a_k - \bar{a})^2}.$$

For example, if the five CNN fold accuracies were $[0.610, 0.567, 0.561, 0.535, 0.626]$, then:

$$\bar{a} = \frac{0.610 + 0.567 + 0.561 + 0.535 + 0.626}{5} = 0.580.$$

This is why I report CNN full augmentation as $58.0\% \pm 3.3\%$.

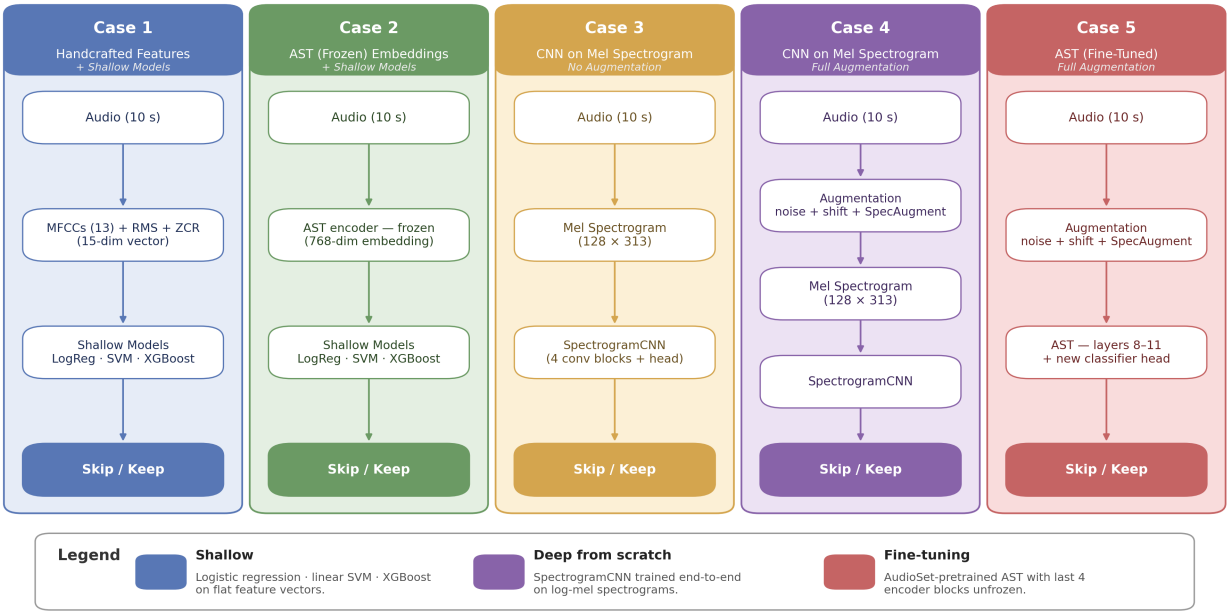

Figure 7: Model comparison framework. The five main audio modeling strategies compare handcrafted features, frozen AST embeddings, CNNs with and without augmentation, and fine-tuned AST. Context-only and fusion models are reported separately in the results table as diagnostic extensions.

7.2 Metrics

Accuracy is easy to interpret, but it can be misleading because the strict matched dataset is not perfectly balanced. The strict matched dataset contains 455 Keep tracks and 714 Skip tracks, so a naive majority-class model can look better than chance by predicting Skip often. For that reason, I report a dummy majority baseline and also use balanced accuracy, F1, and AUC. Balanced accuracy is especially important because it averages performance across classes rather than rewarding a model for mostly predicting the larger class.

Accuracy is:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

F1 is:

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}},$$

where

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}.$$

For example, suppose a Skip-class evaluation gives 10 true Skip predictions, 12 false Skip predictions from Keep songs, and 5 false Keep predictions from Skip songs. Then:

$$\text{Precision}_{\text{Skip}} = \frac{10}{10+12} = 0.455,$$

$$\text{Recall}_{\text{Skip}} = \frac{10}{10+5} = 0.667,$$

$$F1_{\text{Skip}} = \frac{2(0.455)(0.667)}{0.455+0.667} = 0.541.$$

This example shows why F1 is useful: it combines how precise the model is when it predicts Skip with how many true Skip cases it actually finds.

Balanced accuracy is:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right).$$

This metric matters here because a model that predicts most songs as Skip may have acceptable accuracy but poor Keep recall.

The dummy majority baseline is important because the strict dataset has more Skip tracks than Keep tracks. A model that always predicts the majority class can achieve a non-trivial raw accuracy without learning anything about the audio. For that reason, I do not interpret accuracy by itself. A useful model should improve balanced accuracy, F1, or AUC relative to this dummy baseline, not only raw accuracy.

8 Model Selection and Mathematical Explanation

8.1 Binary Cross-Entropy Loss

For models that output a probability, the predicted probability is:

$$\hat{p}_i = P(y_i = 1 \mid x_i),$$

where x_i is the audio representation for track i and y_i is the true label. The binary cross-entropy loss is:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right].$$

This loss is useful because it punishes confident wrong predictions more than uncertain wrong predictions. Suppose a song is truly Skip, so $y_i = 1$. If the model predicts $\hat{p}_i = 0.80$, the loss is:

$$-\log(0.80) = 0.223.$$

If the model predicts $\hat{p}_i = 0.20$, the loss is:

$$-\log(0.20) = 1.609.$$

So the model pays a much larger penalty for being confidently wrong.

8.2 Logistic Regression

Logistic regression takes a feature vector $x_i \in \mathbb{R}^d$ and computes:

$$z_i = w^\top x_i + b.$$

Then it converts the score into a probability:

$$\hat{p}_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}.$$

In this project, x_i might be a 15-dimensional handcrafted audio vector, a 768-dimensional AST embedding, or a concatenated multimodal vector. The weights w represent how much each feature contributes to the skip score. A positive weight increases the predicted skip probability when that feature is high. A negative weight decreases it. The bias b is the baseline tendency to predict Skip before looking at the features.

A concrete toy example:

$$x_i = [0.8, 0.3, 1.2], \quad w = [1.0, -0.5, 0.25], \quad b = -0.1.$$

Then:

$$z_i = (1.0)(0.8) + (-0.5)(0.3) + (0.25)(1.2) - 0.1 = 0.8 - 0.15 + 0.3 - 0.1 = 0.85.$$

The skip probability is:

$$\hat{p}_i = \sigma(0.85) = \frac{1}{1 + e^{-0.85}} = 0.700.$$

With a threshold of 0.50, this would be predicted as Skip.

8.3 Support Vector Machine with RBF Kernel

The RBF SVM was important because shallow models performed surprisingly well. The RBF kernel is:

$$K(x_i, x_j) = \exp(-\gamma ||x_i - x_j||^2).$$

Here, γ controls how quickly similarity decays with distance. If γ is large, only very close points are treated as similar. If γ is small, even farther points can influence each other.

For a concrete example, suppose two 2-dimensional feature vectors are:

$$x_i = [0, 0], \quad x_j = [1, 1],$$

and $\gamma = 0.5$. Then:

$$||x_i - x_j||^2 = (0-1)^2 + (0-1)^2 = 2.$$

So:

$$K(x_i, x_j) = \exp(-0.5 \cdot 2) = e^{-1} = 0.368.$$

This means the model treats these two examples as moderately similar. In my project, the real vectors are 15-dimensional handcrafted features or 768-dimensional AST embeddings, but the idea is the same. The SVM uses similarities to build a nonlinear boundary between Skip and Keep songs.

8.4 K-Nearest Neighbors on AST Embeddings

K-nearest neighbors was important because the best strict-label audio-only model was Frozen AST embeddings + KNN with $k = 11$. Unlike logistic regression, KNN does not learn a weight vector. Instead, it stores the training examples and classifies a new song by comparing it to nearby songs in the feature space.

For an input song with embedding x, KNN finds the k training embeddings closest to x. I use Euclidean distance:

$$d(x_i, x_j) = ||x_i - x_j||_2 = \sqrt{\sum_{\ell=1}^{d} (x_{i\ell} - x_{j\ell})^2}.$$

In the AST case, each vector has dimension $d = 768$. Each coordinate is one learned feature from the pretrained audio transformer. The KNN prediction is the majority vote among the nearest neighbors:

$$\hat{y}(x) = \text{majority vote} \left(y_{(1)}, y_{(2)}, \ldots, y_{(k)} \right),$$

where $y_{(1)}, \ldots, y_{(k)}$ are the labels of the k nearest training songs.

A concrete example: if $k = 11$ and the 11 nearest AST embeddings contain 7 Skip tracks and 4 Keep tracks, the model predicts Skip. If they contain 6 Keep tracks and 5 Skip tracks, it predicts Keep. This works well when the pretrained embedding space places acoustically or behaviorally similar songs near each other. In my results, this was stronger than a learned linear boundary, which suggests that local neighborhoods in AST space captured useful skip-related structure.

8.5 CNN on Mel Spectrograms

The CNN architecture is shown in fig. 8. I implement the CNN in PyTorch, which is useful here because it gives direct control over custom datasets, training loops, augmentation, and GPU acceleration (Paszke et al., 2019). The input is a tensor of shape:

$$1 \times 128 \times 313.$$

The first convolutional block outputs 16 channels and reduces the time-frequency dimensions through pooling. Later blocks output 32, 64, and 128 channels. The final feature vector has dimension 128, and the classifier head maps it to 2 logits: one for Keep and one for Skip.

A convolutional layer computes:

$$O_{i,j,m} = \sum_{u=1}^{k} \sum_{v=1}^{k} \sum_{c=1}^{C_{\text{in}}} X_{i+u-1,j+v-1,c} W_{u,v,c,m} + b_m.$$

In this equation:

- X is the input spectrogram or previous feature map.
- W is a learned filter.
- b_m is a learned bias for output channel m.
- $O_{i,j,m}$ is the activation at time-frequency location (i, j) in channel m.

A toy numerical example helps. Suppose a 2×2 spectrogram patch is:

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

and a 2×2 filter is:

$$W = \begin{bmatrix} 0.5 & -0.25 \\ 0.1 & 0.2 \end{bmatrix}$$

with bias $b = 0.1$. The convolution output at this patch is:

$$(1)(0.5) + (2)(-0.25) + (3)(0.1) + (4)(0.2) + 0.1 = 0.5 - 0.5 + 0.3 + 0.8 + 0.1 = 1.2.$$

If this activation passes through ReLU, it stays 1.2. In the real model, the filters are 3×3, and the model learns many of them. One filter might respond to vertical stripes that represent repeated beats, while another might respond to low-frequency energy or sudden changes.

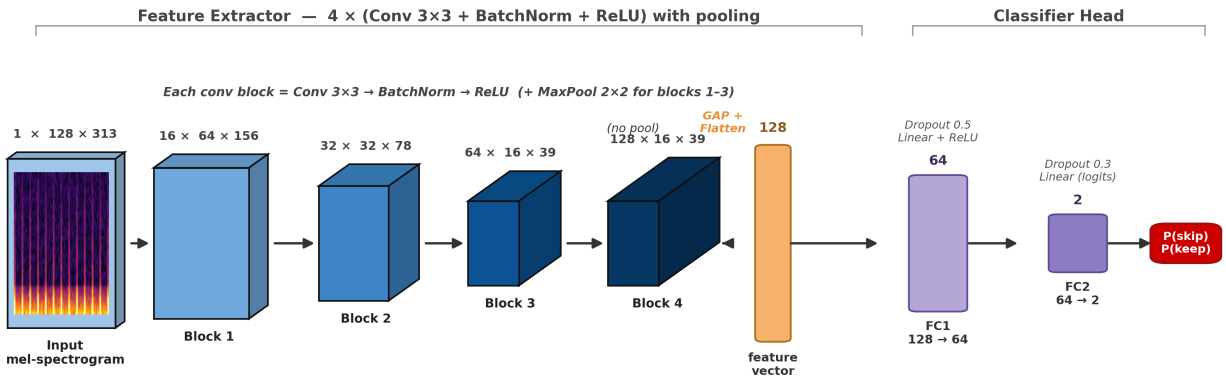

Figure 8: SpectrogramCNN architecture. A one-channel log-mel spectrogram passes through four convolutional blocks, global average pooling, and a two-layer classifier head. The model is intentionally small because the strict dataset is small for deep learning.

8.6 Fine-Tuned Audio Spectrogram Transformer

The AST architecture is shown in fig. 9. AST treats the spectrogram like an image made of patches. Each patch becomes a vector token, and the transformer uses self-attention to let each patch compare itself to other patches. This follows the Audio Spectrogram Transformer approach, which adapts vision-transformer-style patch modeling to audio spectrograms (Gong, Chung, and Glass, 2021).

Scaled dot-product attention is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

The matrices mean:

- Q is the query matrix: what each patch is looking for.
- K is the key matrix: what each patch offers for matching.
- V is the value matrix: the information that gets mixed after attention weights are computed.
- d_k is the key dimension, used to scale dot products so they do not become too large.

A small numerical example: suppose one query compares to two keys and produces scores $[2, 0]$. The softmax is:

$$\text{softmax}([2, 0]) = \left[\frac{e^2}{e^2 + e^0}, \frac{e^0}{e^2 + e^0}\right] = [0.881, 0.119].$$

This means the first patch receives 88.1% of its mixed information from the first value and 11.9% from the second. In a song spectrogram, this allows the model to connect patterns across time and frequency, such as relating an early drum onset to a later harmonic texture.

The fine-tuning strategy freezes encoder layers 0–7 and unfreezes layers 8–11 plus a new classifier head. This follows the transfer-learning idea that lower layers can preserve general audio

representations while later layers adapt to the new target task (Pan and Yang, 2010). The classifier head is:

$$\mathbb{R}^{768} \to \mathbb{R}^2.$$

The 768-dimensional vector is a learned clip embedding after mean pooling over patch tokens. The two outputs are the Keep and Skip logits.

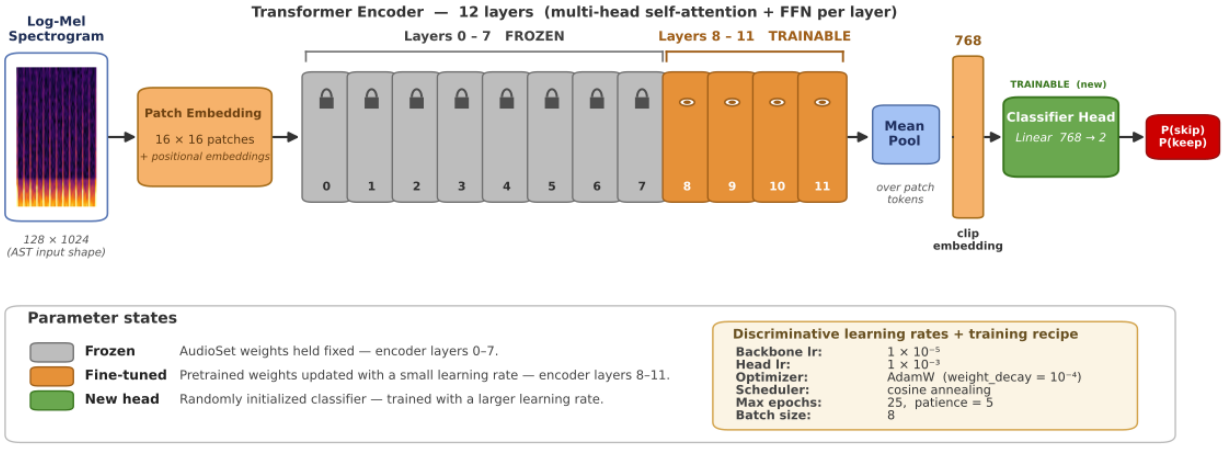

Figure 9: Fine-tuned AST architecture. The pretrained AST keeps encoder layers 0–7 frozen and fine-tunes layers 8–11 plus a new two-class classifier head. This uses transfer learning while limiting the number of trainable parameters.

8.7 Context-Only Baseline

The context-only baseline uses the same contextual variables as the fusion model, but removes all audio features. If c_i is the context vector for track i, then the model predicts:

$$\hat{p}_i = P(y_i = 1 \mid c_i).$$

This baseline is necessary because it tells me whether the fusion model is actually using audio or mostly relying on listening context. If context-only performs close to fusion, then the main gain comes from behavior and listening history rather than from the 10-second song intro. If fusion clearly beats context-only, then the audio representation adds complementary information.

8.8 Multimodal Fusion: Audio Features Plus Context

The fusion models test whether audio becomes more predictive when paired with listening context. This is not the same as the pure audio-only task. In the audio-only task, the model sees only the first 10 seconds of sound. In the fusion task, the model also sees context features such as platform, shuffle behavior, time of day, and artist or album familiarity.

For the CNN fusion model, I first use the trained CNN as a feature extractor. Let x_i be the log-mel spectrogram for song i. The CNN maps it to a 128-dimensional penultimate feature vector:

$$h_{\text{CNN}}(x_i) \in \mathbb{R}^{128}.$$

Let c_i be the context vector for the same song. The fusion input is the concatenation:

$$z_i = [h_{\text{CNN}}(x_i), c_i].$$

Here, z_i contains both what the song sounds like and information about how I tend to listen. A classifier then predicts:

$$\hat{p}_i = P(y_i = 1 \mid z_i).$$

For example, suppose $h_{\text{CNN}}(x_i)$ has 128 audio features and the processed context vector has 10 features after scaling and one-hot encoding. Then:

$$z_i \in \mathbb{R}^{138}.$$

This means the classifier receives 138 total inputs: 128 learned audio features and 10 contextual features. If fusion performs much better than audio alone, the correct interpretation is not that the first 10 seconds completely solve the task. The correct interpretation is that skip prediction needs information outside the waveform. This is why I treat fusion as a diagnostic extension rather than as the main answer to the audio-only question.

This also creates one important caution: the fusion model is not a pure new-song predictor. Some context features come from historical listening behavior, so they are available for songs already present in my archive but not necessarily for a completely new song with no listening history.

9 Training Algorithm

The full training pipeline is summarized below. This is the algorithmic version of the code in the notebook.

1. Load Spotify streaming history JSON files.
2. Convert event-level plays into Skip/Keep labels using 30-second and 60-second thresholds.
3. Aggregate events to track-level skip rates.
4. Build loose and strict track labels.
5. Match tracks to Deezer previews and download/cache audio.
6. Preprocess audio: resample to 16 kHz, convert to mono, trim or pad to 10 seconds, normalize.
7. Build feature representations:
   (a) 15-dimensional handcrafted feature vectors,

(b) 768-dimensional frozen AST embeddings,

(c) $1 \times 128 \times 313$ log-mel spectrogram tensors,

(d) contextual listening features.

8. Split the data into training and holdout sets.
9. Run cross-validation on the training split for each model family.
10. Train final finalist models.
11. Evaluate once on the holdout split.
12. Report accuracy, balanced accuracy, F1, AUC, confusion matrices, and calibration plots.

The code includes a `FAST_MODE` switch. In fast mode, fewer folds and epochs are used for debugging. In final mode, the notebook runs fuller cross-validation and longer training. The neural-network models are implemented in PyTorch, while the shallow baselines and cross-validation utilities use `scikit-learn` (Paszke et al., 2019; Pedregosa et al., 2011). This matters for #MLCode because it makes the notebook practical to debug while still preserving a complete final run.

10 Results

10.1 Shallow Models

The shallow baselines were not throwaway models. In fact, they performed competitively. On the strict set, Frozen AST + KNN achieved 61.8% ± 2.4% accuracy, while Simple 15-dimensional features + KNN achieved 56.7% ± 2.8%. This suggests the pretrained AST embedding does contain useful audio information. It also suggests that a nonlinear neighborhood-based classifier can work well when the dataset is not very large.

10.2 CNN Results

The CNN results are shown in fig. 10. On loose labels, the CNN without augmentation reached 54.3% ± 4.0%, and the CNN with full augmentation reached 54.4% ± 3.4%. On strict labels, the CNN without augmentation reached 60.1% ± 1.9%, while the CNN with full augmentation reached 58.0% ± 3.3%.

This is a useful result, but not a clean victory for augmentation. The strict-label CNN without augmentation slightly outperformed the augmented CNN. The stronger interpretation is that strict labels helped the CNN, but augmentation did not consistently improve mean accuracy in the final run.

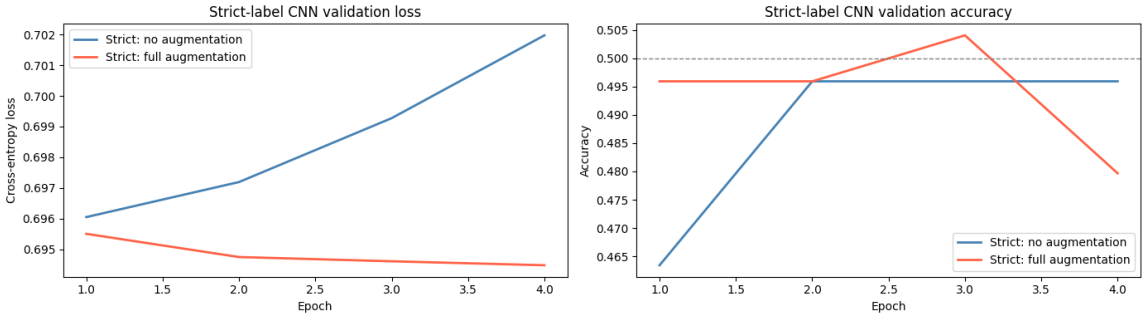

Figure 10: CNN validation accuracy across epochs. The left panel shows loose-label CNN behavior and the right panel shows strict-label CNN behavior. Strict labels make the CNN more competitive, but the augmented model does not clearly dominate the non-augmented model.

10.3 Augmentation Ablation

The augmentation ablation is shown in fig. 11. The final notebook's ablation result indicates that no augmentation condition clearly dominates. The highest mean in the plotted two-fold ablation is no augmentation at 51.2%, while full augmentation is 49.6%.

This matters because it prevents an overclaim. I should not say "augmentation solved the problem." A more accurate claim is that augmentation was tested carefully and made the training pipeline more flexible, but the evidence for an accuracy gain was weak in this particular dataset.

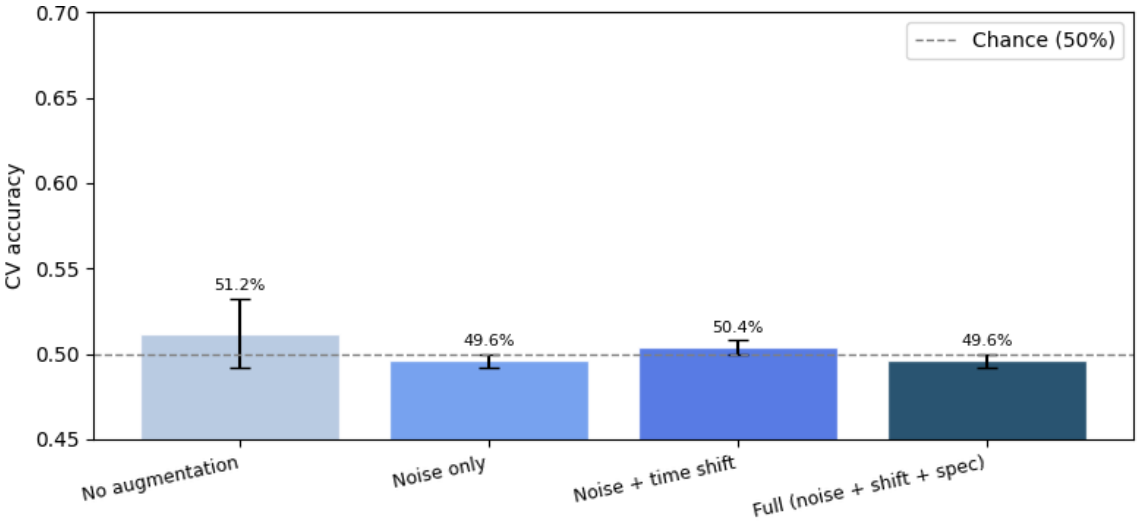

Figure 11: Augmentation ablation for the CNN on strict labels. The ablation compares no augmentation, noise only, noise plus time shift, and full augmentation. The differences are small and close to chance, so the conclusion should be cautious.

10.4 Fine-Tuned AST Results

Fine-tuned AST reached 56.0% validation accuracy, 56.4% balanced accuracy, and 0.610 AUC in the fixed-validation pilot. This is a meaningful #MLFlexibility component because it adapts a pretrained transformer to a personal audio classification task. However, I treat it as a pilot rather than the main winner because it was not run as full 5-fold cross-validation in the same way as the lighter models.

10.5 Context-Only and Fusion Results

The context-only and fusion models test whether the audio-only task is missing behavioral information. This comparison is necessary because a fusion model may improve either because audio and context are complementary, or because context alone already explains most of the skip behavior. For that reason, I report context-only and fusion rows side by side rather than treating fusion as a direct answer to the 10-second audio question.

The correct control here is context-only performance. If context-only is close to fusion, then context explains most of the gain. If fusion clearly beats context-only, then audio and context are complementary. For that reason, I include context-only rows in the final comparison rather than treating fusion as a direct audio-only model.

Representation	Modality	Classifier	CV Acc	CV Bal Acc	CV F1	CV AUC
AST embeddings	Context-only	Logistic Regression	75.1% ± 0.8%	74.6% ± 0.6%	0.79 ± 0.01	0.817 ± 0.014
CNN penultimate features	Context-only	Logistic Regression	75.1% ± 0.8%	74.6% ± 0.6%	0.79 ± 0.01	0.817 ± 0.014
AST embeddings	Context-only	MLP	76.4% ± 1.7%	73.5% ± 1.6%	0.82 ± 0.02	0.809 ± 0.021
CNN penultimate features	Context-only	MLP	76.4% ± 1.7%	73.5% ± 1.6%	0.82 ± 0.02	0.809 ± 0.021
CNN penultimate features	Fusion	Logistic Regression	73.4% ± 2.5%	73.1% ± 2.3%	0.77 ± 0.03	0.790 ± 0.026
CNN penultimate features	Fusion	MLP	72.9% ± 2.4%	71.4% ± 2.8%	0.78 ± 0.02	0.778 ± 0.031
AST embeddings	Fusion	Logistic Regression	66.4% ± 3.0%	65.0% ± 2.7%	0.72 ± 0.04	0.706 ± 0.024
AST embeddings	Fusion	MLP	66.3% ± 5.1%	63.9% ± 4.6%	0.73 ± 0.05	0.715 ± 0.028

Table 3: Context-only versus fusion diagnostic table. This table checks whether the gain comes mostly from context alone or from the combination of audio and context.

10.6 Holdout Results

Because the full holdout run is computationally expensive, I treat the cross-validation tables as the main model-comparison evidence in this written version. The notebook implements the full strict holdout evaluation, and I include the notebook in the accompanying Drive folder once the run is complete. The figures below are therefore used as diagnostic checks, not as the final model-ranking evidence.

The diagnostic holdout figures are still useful because they show how audio-only finalists can fail. In particular, they reveal whether models are learning both classes or mostly predicting the majority class. This supports the main conclusion from cross-validation: audio-only models contain

limited signal, while context-based models explain much more of the task.

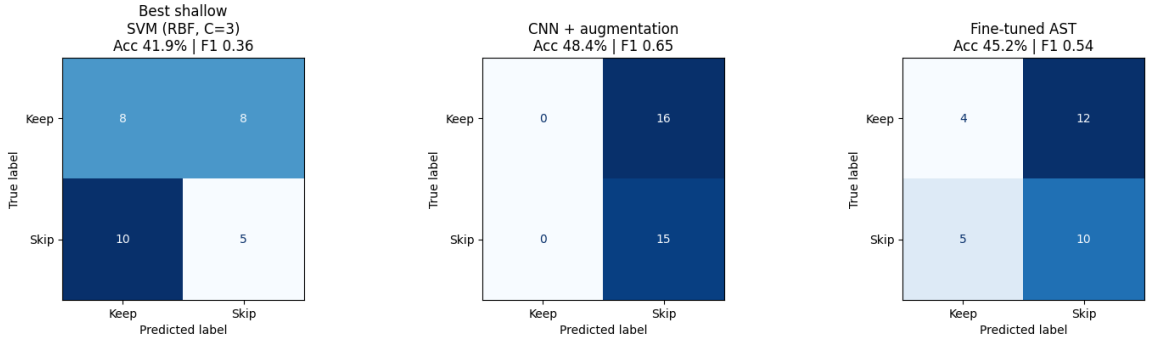

Figure 12: Holdout confusion matrices for audio-only finalists. These matrices compare the main audio-only finalist models on the shared diagnostic holdout subset. I use this figure as a qualitative sanity check, not as the main model-ranking evidence.

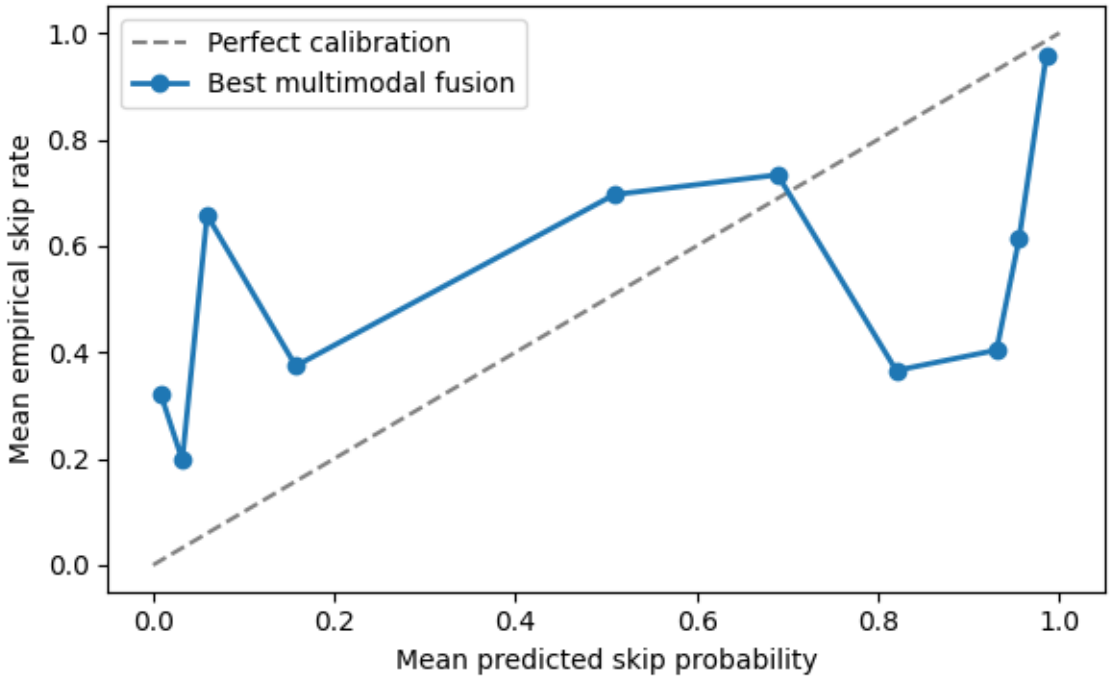

Figure 13: Calibration plot for the cached fusion probability model. The dashed diagonal shows perfect calibration. This figure checks whether predicted skip probabilities match empirical skip rates, not just whether the hard class labels are correct. I use it as a diagnostic plot rather than as evidence that fusion is the strongest model.

11 Discussion

The most important finding is not that one big model wins. The most important finding is that the audio-only task contains weak but real signal, while listening context explains much more of the

prediction task. Frozen AST embeddings, scratch CNNs, and fine-tuned AST all found some signal, but the gains were modest: the best audio-only model reached 61.8% cross-validation accuracy, with balanced accuracy below 60%. That is not strong enough to claim that 10-second audio alone solves skip prediction.

The context-only and fusion comparison changes the interpretation. Context-only Logistic Regression outperformed the best fusion model in the final cross-validation table. This suggests that the strongest signal is not in the audio representation, but in behavioral context. In other words, the model does better when it knows how and when I tend to listen than when it only hears the song intro.

This makes sense. Audio models can hear timbre, rhythm, loudness, and texture. They cannot know whether I usually listen to an artist while walking, whether a song appears in a playlist I skip through quickly, or whether I am replaying a familiar album. Those contextual variables are not failures of the audio model. They are missing information.

This project also shows why strict labeling matters. The loose label includes many songs whose skip rate is near 0.50. Those songs are not clearly Skip or Keep. The strict label removes them, which makes the learning target more honest. Even then, the model does not become easy. That suggests the remaining noise is not only label noise. It is also behavioral complexity.

12 Limitations

1. **The labels are inferred, not directly observed.** A short play is treated as a skip, but some short plays may come from interruptions or app behavior.
2. **The audio source and label source differ.** Spotify provides the behavior; Deezer provides the preview. Version mismatches can add noise.
3. **The strict task is smaller.** Strict labels reduce noise but also reduce sample size.
4. **The audio window is short.** Some songs become recognizable after the first 10 seconds.
5. **Fine-tuned AST is only a pilot.** It is valuable for transfer learning, but it was not computationally feasible to run full 5-fold fine-tuning for every setting.
6. **Fusion is not a pure audio model.** The fusion result is useful because it diagnoses missing information, but it should not be interpreted as evidence that 10-second audio alone predicts skips. Some context features also depend on past listening behavior, so the fusion model is less applicable to completely new songs with no history.
7. **The holdout diagnostics are limited.** The notebook implements the full 234-track holdout evaluation, but this written version relies mainly on cross-validation because the final run is computationally expensive. The cached holdout confusion matrices should be read as qualitative diagnostics, not as the final ranking of model families.

13 Future Work

The best next step is not simply adding more epochs. More epochs would help only if validation loss were still improving. The better improvements are:

- Use longer windows, such as 20 or 30 seconds.
- Use a music-specific pretrained model instead of a general AudioSet AST.
- Model listening sessions, not just isolated songs.
- Use class weights and all matched clips rather than discarding data to balance classes.
- Improve version matching between Spotify and Deezer.
- Develop a stronger multimodal architecture that combines audio and context earlier.

14 Conclusion

This project asked whether the first 10 seconds of a song can predict whether I will skip it. The audio-only answer is: partially, but not reliably. Frozen AST embeddings with KNN produced the strongest audio-only cross-validation result at 61.8% ± 2.4% accuracy, and the CNN without augmentation reached 60.1% ± 1.9%. These results show that song intros contain some predictive signal. However, the balanced accuracy and AUC values are modest, and the audio-only holdout diagnostics are unstable. So I should not claim that audio alone solves the task.

The stronger result comes from listening context. Context-only Logistic Regression reached 75.1% ± 0.8% accuracy, 74.6% ± 0.6% balanced accuracy, and 0.817 ± 0.014 AUC in strict-label cross-validation. The best fusion model did not clearly outperform this context-only baseline. I interpret this not as a failure of the project, but as the main finding: my skip behavior is more strongly explained by listening context than by the first 10 seconds of audio alone.

This also makes the audio-only result easier to interpret. The song intro contains some signal, but it is not the main driver of prediction in this dataset.

The project succeeds as a machine learning investigation because it builds a full pipeline from personal data to labels, audio preprocessing, feature extraction, model comparison, augmentation, fine-tuning, fusion, and evaluation. The key result is not a perfect predictor. The key result is a clearer understanding of the problem: personal skip behavior is not only in the song intro. It is in the interaction between the song, the listener, and the listening situation.

AI Use Statement

I used AI tools (Claude and ChatGPT) during the later stages of the assignment to debug API and formatting issues, to check whether the notebook covered the required assignment sections, and to help reduce redundant writing. I also used AI support when I got stuck with Spotify preview access and Deezer matching, and again when organizing the final LaTeX version. The data source,

modeling decisions, code execution, interpretation of the results, and final claims were reviewed and edited by me.

References

- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- Davis, S., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 28*(4), 357–366.
- Gong, Y., Chung, Y.-A., & Glass, J. (2021). AST: Audio Spectrogram Transformer. *Interspeech*.
- Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization.
- McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and music signal analysis in Python. *Proceedings of the 14th Python in Science Conference*.
- Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering, 22*(10), 1345–1359.
- Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, E., Cubuk, E. D., & Le, Q. V. (2019). SpecAugment: A simple data augmentation method for automatic speech recognition. *Interspeech*.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., et al. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of EMNLP 2020: System Demonstrations*.

A Appendix A: Additional Diagnostic Figures

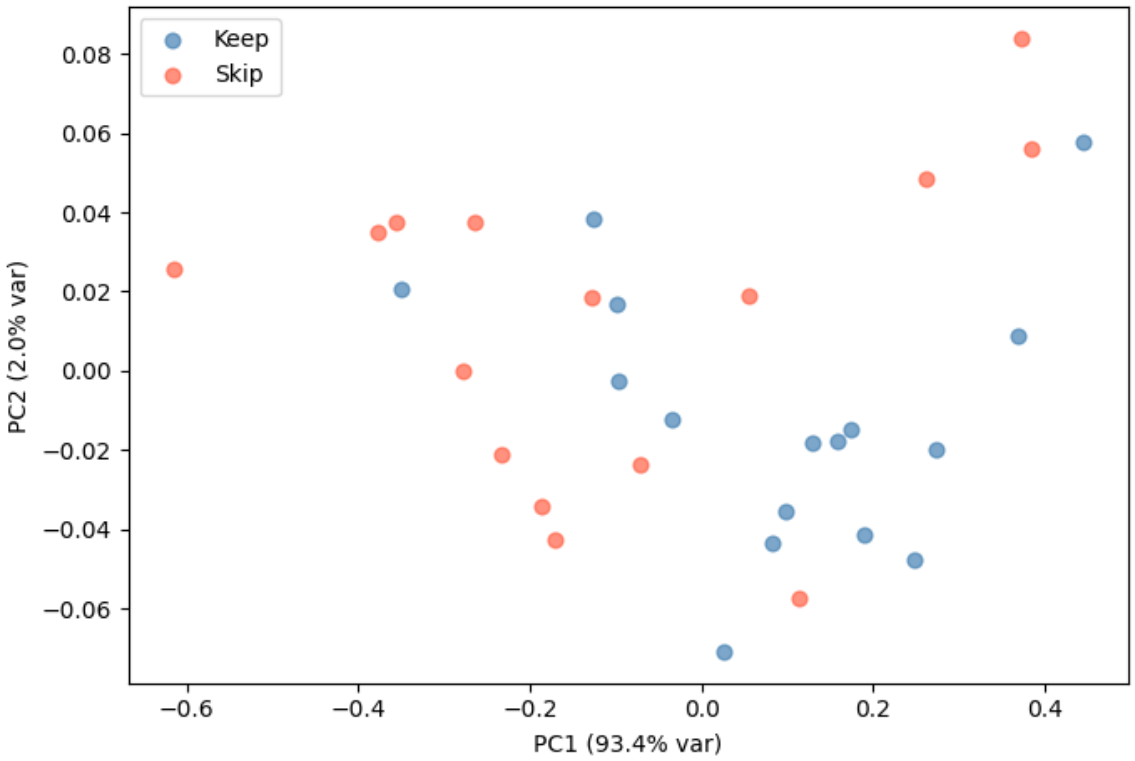

Figure 14: PCA diagnostic for CNN features. This plot projects the final 128-dimensional CNN penultimate features into two dimensions. It is not used for classification, but it gives a sanity check of whether the CNN representation contains visible structure.

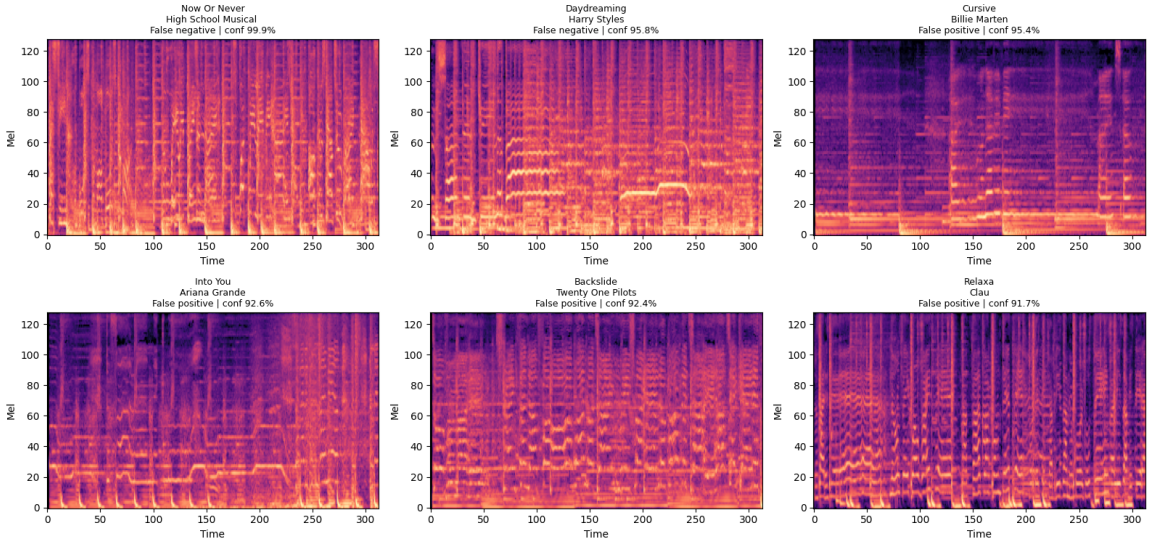

Figure 15: Misclassified examples from the cached fusion diagnostic. Each panel shows a spectrogram for a holdout song that the fusion model misclassified. These examples are useful for qualitative diagnosis, but I keep them in the appendix because the main results are better summarized by the cross-validation tables.

B Appendix B: Code Organization and Reproducibility

The full implementation is provided in the accompanying Google Drive folder. The folder includes the notebook, Spotify history files, and generated outputs. I keep the full code outside this paper so the write-up remains readable, while the implementation remains inspectable.

Full notebook and the Spotify History Files are in this Google Folder: https://drive.google.com/drive/folders/1B5NNnKhcxJclURle3n9gQ6lcpblRIycs?usp=drive_link

Code block	Purpose
Data loading	Loads six Spotify JSON files, parses timestamps, removes private columns, and filters to music tracks.
Label construction	Converts play events into Skip/Keep labels, aggregates repeated plays into track-level skip rates, and creates loose and strict labels.
Deezer matching	Searches Deezer by title and artist, filters mismatched versions, downloads previews, and caches matched audio.
Audio preprocessing	Loads audio, resamples to 16 kHz, converts to mono, trims or pads to 10 seconds, and creates log-mel spectrograms.
Feature extraction	Computes handcrafted features, frozen AST embeddings, CNN penultimate features, and context features.
Modeling	Runs shallow baselines, CNN cross-validation, augmentation ablation, fine-tuned AST pilot, and multimodal fusion models.
Evaluation	Computes accuracy, balanced accuracy, F1, AUC, confidence intervals, confusion matrices, calibration plots, and error diagnostics.

Table 4: Code organization in the accompanying notebook.

The notebook also includes a `FAST_MODE` switch for debugging and a final-mode configuration for the complete run. This was important because CNN and AST experiments are computationally expensive, while the pipeline still needed to be testable end to end.