EUCLID: Multi-Head Binary Classification System for Synthetic Data Detection and Audio Authentication

Sabian Hibbs BSc, MSc
Chief Technology Officer
Uhmbrella Ltd.
Email: sabian@uhmbrella.io

***Abstract*—The proliferation of synthetically generated content presents a formidable challenge within contemporary machine learning paradigms, particularly in the domain of artificially generated audio. This white paper delineates EUCLID (Enhanced Utility for Classification and Identification of Data), an innovative multi-head binary classification architecture that discriminates between authentic and synthetic data through an ensemble of specialized neural network substructures. The proposed framework integrates sophisticated probabilistic formulations, a novel asymmetric output aggregation methodology, and a meticulously engineered data processing pipeline. This manuscript details the theoretical foundations of the EUCLID model with mathematics, clearly explains each computational module, and concludes with performance projections and future research directions.**

I. Introduction

Detecting synthetic data, especially artificial audio, is a significant challenge in modern machine learning. This manuscript delineates a multi-head binary classification architecture designed for discriminating between authentic and synthetically generated data specimens. The framework integrates multiple neural network substructures ("heads") whose outputs undergo a specialized aggregation protocol engineered to enhance detection sensitivity. This sophisticated approach incorporates a rigorous probabilistic formalism, an ensemble classification methodology with a novel asymmetric output integration strategy, and a comprehensive data transformation pipeline. The system's modular design—encompassing all operational stages from preprocessing to inference—facilitates adaptability and renders it appropriate for inclusion in technical documentation or project repositories.

The organization of this discourse proceeds as follows: We commence with a formal mathematical exposition of the classification model, elucidating the probabilistic computation framework, loss function formulation, decision boundary determination methodology, and the theoretical justification for the distinctive averaging of "authentic" logits across multiple neural substructures. Subsequently, we provide exhaustive explications of each computational module within the implementation pipeline, articulating how the constituent scripts collectively form an integrated detection system:

- `file_renamer.py` – ensures consistent unique filenames for audio data.
- `audio_converter.py` – normalizes audio format and sampling.
- `audio_augment.py` – generates augmented audio examples to expand the training set.
- `audio_segmenter.py` – chops audio files into fixed-length segments.
- `dataset_manager.py` – splits data into training and testing sets by class.
- `file_manager.py` – checks and fixes any overlap between training and testing sets (to prevent data leakage).
- `submodel_trainer.py` – trains individual classification sub-models (the "heads").
- `model_merger.py` – merges multiple trained sub-models into a unified multi-head ensemble model.
- `inference_runner.py` – runs the merged model on new data for inference, producing detection results.

Finally, we examine the performance characteristics of our multi-head classification architecture. The analysis includes projected accuracy metrics and computational efficiency assessments for the system's ensemble-based design and modular processing pipeline. We highlight how the combination of specialized detection heads contributes to overall system robustness and sensitivity in distinguishing between authentic and synthetic audio specimens.

II. Multi-Head Classification Model: Mathematical Framework

The core of the system is a binary classifier implemented as an ensemble of neural network sub-models (multiple "heads"). Each sub-model is a deep neural network that independently attempts to classify an audio sample as Real (authentic) or Synthetic (fake). By combining several sub-models, the system aims to improve robustness and detection accuracy through

ensemble learning. In this section, we detail the model architecture and derive its probability estimates, loss function formulation, decision strategy, and explain the reasoning for the unique averaging of real logits.

A. Model Architecture and Probability Computation

Each sub-model instantiates a convolutional neural architecture that operates on spectrotemporal representations of acoustic signals. In our implementation, individual sub-models utilize a *pretrained convolutional backbone* (specifically, a residual network architecture such as ResNet-18) for extracting hierarchical feature representations from input spectrograms, followed by a domain-specific classification module that projects these representations onto a binary decision space. Formally, given an input representation x (a log-mel spectrogram derived from an audio segment), each sub-model f_i (parameterized by θ_i) computes a two-dimensional projection into logit space:

$$z_i = f_i(x; \theta_i) = (z_{i,\text{real}},\ z_{i,\text{synthetic}}), \tag{1}$$

where $z_{i,\text{real}}$ is the logit (unnormalized score) that the sample is real, and $z_{i,\text{synthetic}}$ is the logit that the sample is synthetic. (Some implementations may swap the order; what matters is consistent interpretation of each index.)

To convert logits into class probabilities for a given sub-model, we apply the softmax function. For sub-model i, the predicted probability of the sample being real or synthetic is:

$$p_{i,\text{real}} = \frac{e^{z_{i,\text{real}}}}{e^{z_{i,\text{real}}} + e^{z_{i,\text{synthetic}}}}, \tag{2}$$

$$p_{i,\text{synthetic}} = \frac{e^{z_{i,\text{synthetic}}}}{e^{z_{i,\text{real}}} + e^{z_{i,\text{synthetic}}}}, \tag{3}$$

which ensures $p_{i,\text{real}} + p_{i,\text{synthetic}} = 1$. This is a special case of the softmax for $C = 2$ classes (binary classification), where we normalize the two logits to obtain a valid probability distribution. In general, for a classifier with C classes and logits $z = (z_1, z_2, \ldots, z_C)$, the softmax defines

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}. \tag{4}$$

Here $C = 2$, and it simplifies as above.

Our system combines N such sub-models. We construct a multi-head ensemble model $F(x)$ that aggregates the outputs of all sub-models. The ensemble's forward pass collates the logits from each sub-model:

- For each sub-model $i = 1, \ldots, N$, compute $z_i = (z_{i,\text{real}}, z_{i,\text{synthetic}})$.
- Separate the real and synthetic components: collect all $z_{i,\text{real}}$ into a set $\{z_{1,\text{real}}, \ldots, z_{N,\text{real}}\}$ and all synthetic logits $\{z_{1,\text{synthetic}}, \ldots, z_{N,\text{synthetic}}\}$.
- The ensemble real logit is defined as the average of all real logits:

$$z_{\text{ensemble, real}} = \frac{1}{N} \sum_{i=1}^{N} z_{i,\text{real}}. \tag{5}$$

- All the synthetic logits are kept separate (each sub-model contributes its own synthetic logit). Thus, the ensemble output is a vector of length $N + 1$:

$$F(x) = (z_{1,\text{synthetic}},\ z_{2,\text{synthetic}},\ \ldots,\ z_{N,\text{synthetic}},\ z_{\text{ensemble, real}}). \tag{6}$$

This ensemble output configuration effectively generates a vector comprising N distinct detection scores corresponding to synthetic characteristics (with each score derived from an individual classification head) and a singular consolidated authenticity score.

For probabilistic interpretation of the ensemble's output representation, one could theoretically employ softmax normalization across these $N + 1$ components. Such an approach would conceptualize the classification paradigm as an $(N + 1)$-class taxonomic problem, wherein each synthetic detection head constitutes an independent category alongside a unified authentic class. However, given that our fundamental classification objective remains dichotomous (distinguishing authentic from synthetic specimens), we typically operationalize the ensemble output through comparative analysis between the maximal synthetic detection score and the consolidated authenticity score (elaborated in the Decision Thresholds subsection). When probabilistic quantification becomes necessary, one might formulate the ensemble's assessment of a specimen's authenticity as:

$$p_{\text{ensemble, real}} = \frac{e^{z_{\text{ensemble, real}}}}{e^{z_{\text{ensemble, real}}} + \sum_{i=1}^{N} e^{z_{i,\text{synthetic}}}}, \tag{7}$$

$$p_{\text{ensemble, synthetic}} = 1 - p_{\text{ensemble, real}}. \tag{8}$$

In practice, the ensemble's decision rule (described later) can circumvent explicitly computing this combined probability by using logits directly.

B. Loss Function for Training

Each constituent neural substructure undergoes training within a dichotomous classification paradigm employing the cross-entropy loss functional—a canonical optimization criterion for taxonomic machine learning architectures. Throughout the training regimen, we designate "Authentic" as categorical designation 0 and "Synthetic" as categorical designation 1 (or the inverse, contingent upon consistent implementation). The training corpus comprises input representations x accompanied by veridical annotations $y \in \{0, 1\}$ (where 0 signifies authentic and 1 denotes synthetic).

For an arbitrary neural substructure yielding output logits $z = (z_{\text{real}}, z_{\text{synthetic}})$ with corresponding probabilistic

distributions $p = (p_{\text{real}}, p_{\text{synthetic}})$ as previously formalized, and given a one-hot encoded ground truth vector $y = (y_{\text{real}}, y_{\text{synthetic}})$ (manifesting as $(1, 0)$ for authentic specimens or $(0, 1)$ for synthetic specimens), the cross-entropy loss functional is expressed as:

$$L = -\Big[y_{\text{real}} \log(p_{\text{real}}) + y_{\text{synthetic}} \log(p_{\text{synthetic}})\Big]. \quad (9)$$

This formula is a special case of the general cross-entropy for C classes:

$$L(\theta) = -\sum_{i=1}^{C} y_i \log p_i. \quad (10)$$

Here $C = 2$, and effectively if the sample is real (so $y_{\text{real}} = 1, y_{\text{synthetic}} = 0$), the loss is $-\log(p_{\text{real}})$; if the sample is synthetic, the loss is $-\log(p_{\text{synthetic}})$. Minimizing this loss trains each sub-model to output a high probability for the correct class.

It warrants emphasis that the constituent neural substructures undergo disjoint optimization regimes (utilizing identical training corpora) via the aforementioned loss functional. Rather than pursuing simultaneous optimization of the integrated multi-head architecture, we independently parameterize N distinct classificatory frameworks (potentially implementing heterogeneous stochastic initializations or architectural variations). This optimization protocol employs contemporary gradient-based methodologies—specifically, in our implementation, the AdamW algorithm (an extension of adaptive moment estimation incorporating decoupled weight decay regularization). This optimizer facilitates parameter updates θ through iterative computation of loss gradients, thereby accelerating convergence trajectories within the high-dimensional parameter manifold. Such methodological decisions ensure superior generalization characteristics while mitigating overfitting phenomena through appropriate regularization constraints.

Upon completion of this distributed training paradigm, we persist each substructure's parametric configuration. The subsequent model integration phase (elaborated in forthcoming sections) retrieves these parameterizations and instantiates them within a unified architectural framework. It merits particular attention that this integration procedure eschews additional optimization or fine-tuning of the composite ensemble; the constituent substructures retain their independently optimized states. The ensemble framework exclusively provides a novel forward propagation mechanism that synthesizes their respective inferential outputs.

C. Decision Thresholds and Inference Strategy

During inference with a novel input specimen, the amalgamated multi-head architecture $F(x)$ generates an asymmetric output configuration comprising N synthetic class logits (each derived from a distinct classification substructure) juxtaposed with a singular consolidated authenticity logit (computed as the arithmetic mean across all substructures). The determination of the categorical designation—authentic versus synthetic—necessitates a decision-theoretic framework that establishes appropriate classification boundaries within this $N+1$-dimensional output manifold. Our methodology employs a decisional criterion predicated upon unanimity for authenticity attribution: a specimen receives an authentic designation exclusively when the consolidated authenticity logit exceeds all individual synthetic logits across the entire ensemble. This decisional paradigm operationalizes a conservative classification protocol wherein any substantial indication of synthetic characteristics from any individual substructure suffices to classify the specimen as non-authentic. Mathematically, this corresponds to comparing the averaged authenticity logit against the supremum of the synthetic logit distribution across all classification substructures.

Concretely, let:

- $S_i = z_{i,\text{synthetic}}$ be the synthetic logit from sub-model i.
- $R_{\text{avg}} = z_{\text{ensemble, real}} = \frac{1}{N}\sum_{i=1}^{N} z_{i,\text{real}}$ be the averaged real logit.

Our (decision rule) is:

- Predict "Real" if R_{avg} is greater than all of the synthetic logits S_i (i.e., if

$$R_{\text{avg}} > \max\{S_1, S_2, \ldots, S_N\}). \quad (11)$$

- Otherwise, predict "Synthetic" (the sample is considered fake because at least one head's synthetic score exceeds the real consensus).

This decision rule can be conceptualized as a threshold comparison operation. In essence, the procedure implements a comparative assessment between the maximal synthetic confidence score and the averaged authentic confidence measure. Should any individual classificatory substructure produce a synthetic logit exceeding the consolidated authenticity logit (rendering that S_i greater than the real average), the ensemble framework defaults to a synthetic classification. Conversely, when all synthetic logits are subordinate to the authentic average, this implies a consensus among all substructures regarding the specimen's authenticity, thus warranting a classification of "real."

In implementation contexts, one might operationalize this decisional protocol through the application of an (argmax) operation on the $N+1$ dimensional output vector of $F(x)$. If the argmax corresponds to the averaged authenticity output (the terminal dimension), we designate "Real"; should it correspond to any synthetic output dimension, we designate "Synthetic." This argmax-based protocol is functionally equivalent to the aforementioned threshold comparison, as the maximal element within $\{S_1, \ldots, S_N, R_{\text{avg}}\}$ dictates the categorical designation. (In scenarios where multiple substructures yield elevated synthetic scores, the argmax identifies the predominant

synthetic signal; when all synthetic scores remain below the authenticity score, the authentic classification prevails.) Thus, the decision boundary emerges dynamically from the relative magnitudes of the ensemble's output components.

It merits observation that within conventional single-model binary classification frameworks, practitioners typically employ a probability threshold of 0.5 (corresponding to a logit value of 0) for categorical demarcation between authentic and synthetic (i.e., synthetic designation occurs when $p_{\text{synthetic}} > 0.5$). Within our multi-dimensional output configuration, we have adopted a generalized methodology: we necessitate that p_{real} surpass all individual $p_{\text{synthetic}}$ contributions to warrant an authentic classification. This ensemble decision criterion has been selected to minimize Type II errors (false negatives), ensuring that any indication of synthetic characteristics from any classificatory substructure results in synthetic designation.

D. Rationale for Averaging the Real Logits (Ensemble Strategy)

A distinctive characteristic of our multi-head architecture is the deliberate asymmetric aggregation methodology applied to the logit distribution: we compute the arithmetic mean of authenticity logits across all classification substructures, while preserving the individual synthetic logits in their disaggregated form. This construction instantiates a "unanimity requisite for authenticity" framework within the decision-theoretic paradigm. The epistemological justification for this methodological decision derives from the following theoretical considerations:

Within the context of artificially generated data detection, false negative classifications (failing to identify synthetic specimens) typically incur substantially higher costs than false positive classifications within most operational contexts. Through averaging authenticity logits, we effectively mandate consensus across all substructures as a precondition for authentic classification. Each substructure must produce elevated authenticity scores to maintain a robust mean. Should even a singular substructure exhibit probabilistic uncertainty (manifested as diminished authenticity logit magnitudes), it attenuates the consolidated mean, thereby reducing the probability that the authenticity score will exceed any individual synthetic detection score. Conversely, by maintaining disaggregated synthetic logits, we prevent the dilution of salient synthetic indicators through aggregation—effectively, a singular substructure generating compelling evidence of synthetic origin can supersede contrary assessments from other classification components.

From a probabilistic formalism perspective, the averaging of authenticity logits approximates the multiplication of independent authenticity probabilities across the ensemble. Assuming stochastic independence between authenticity assessments $p_{i,\text{real}}$ across the N constituent substructures, the joint probability of unanimous authenticity classification is expressed as

$$\prod_{i=1}^{N} p_{i,\text{real}}. \tag{12}$$

Taking the logarithm turns that product into a sum:

$$\log \prod_{i=1}^{N} p_{i,\text{real}} = \sum_{i=1}^{N} \log p_{i,\text{real}}. \tag{13}$$

The logarithmic transformation of probability measures corresponds (modulo a constant factor) to their logit representation. Consequently, the summation of logits—or equivalently, their arithmetic mean scaled by the factor $1/N$—for the authentic class effectuates a principled aggregation of evidential support across all classificatory substructures. Conversely, for synthetic detection, our epistemological framework prioritizes the identification of any substantive synthetic indicators, which corresponds formally to maintaining disaggregated synthetic logits and applying a supremum operator.

This architectural configuration resembles consensus-based discriminative ensembles; however, it transcends conventional majoritarianism in favor of a decisional calculus approximating unanimity for authenticity attribution versus single-veto synthetic designation. Such an asymmetric decisional protocol systematically biases classification toward synthetic attribution under conditions of uncertainty. This methodological predisposition proves particularly efficacious in artificially generated content detection paradigms, wherein Type II errors (false negatives) typically incur substantially higher operational costs than Type I errors (false positives), contingent upon application context. Through the averaging of authenticity logits, the framework imposes stringent criteria for authentic classification: *simultaneous deception across all classification substructures becomes a necessary precondition*. Conversely, genuinely authentic specimens, characterized by their intrinsic veracity, will elicit consistent authenticity signals across the ensemble, manifesting as elevated mean authenticity logits juxtaposed with attenuated synthetic detection measures, thereby facilitating correct taxonomic designation.

In summation, the `averaging of authenticity logits` implements a consolidated measure of ensemble confidence in specimen legitimacy, while synthetic detection follows an "evidential sufficiency principle" wherein any substantial synthetic indicator precipitates non-authentic classification. This methodology enhances robustness through dual mechanisms: diminishing sensitivity to individual substructure biases toward authentic attribution, and leveraging ensemble diversity for detecting heterogeneous synthetic artifacts. The approach constitutes an innovative adaptation of ensemble classification methodologies—rather than implementing isotropic probability aggregation, it instantiates asymmetric output

integration optimized for detection sensitivity. Integrated multi-model frameworks demonstrate superior error characteristics relative to their constituent components. Our architectural strategy specifically targets an operational point in receiver operating characteristic space that privileges artificially generated content detection with elevated confidence measures.

III. System Components and Data Pipeline

The multi-head classifier exists within an integrated computational framework comprising meticulously engineered data transformation and parametric optimization modules. This architectural ecosystem represents the culmination of a sophisticated processing pipeline, wherein each constituent script executes a precisely delineated algorithmic function. Such modular decomposition exemplifies contemporary best practices in computational machine learning methodologies: data undergoes sequential preprocessing and augmentation regimens, training and evaluation protocols maintain rigorous epistemological separation, and auxiliary utilities facilitate dataset curation and structural organization. The following explication delineates the functional architecture of each computational module and elucidates their integrative dynamics within the holistic synthetic data detection framework. These processing stages are presented in their logical sequential order, though pragmatic implementation may permit certain parallel execution pathways within the overarching methodological framework.

The computational pipeline commences with a heterogeneous corpus of acoustic data acquired from disparate provenance (comprising both authentic and synthetically generated specimens). These audio artifacts frequently possess arbitrary nomenclature, potentially encapsulating metadata or exhibiting inconsistent taxonomic patterns. The "File Renamer" module's primary function entails the establishment of a standardized nomenclatural paradigm that simultaneously ensures uniqueness and preserves data anonymization protocols. This algorithmic construct conducts a systematic traversal of a specified directory structure (with optional recursive subdirectory exploration) to identify audio artifacts and effect a deterministic transformation of their identifiers.

Functional Architecture: The module implements a cryptographic transformation by computing a SHA-256 cryptographic hash function on each audio artifact's binary representation, subsequently employing a truncated prefix of this digest (typically the initial 16 hexadecimal characters) as the canonical identifier. For instance, an audio document initially designated as `Interview_with_ABC.wav` undergoes transformation to `1f6999ff03018e9a.wav`. The algorithm preserves the original extension demarcator (e.g., .wav, .mp3) while systematically expunging the preceding nomenclature. When executed in recursive traversal mode, the algorithm navigates hierarchical directory structures to apply this transformation comprehensively.

Theoretical Justification: The implementation of content-based cryptographic hashing facilitates the prevention of nomenclatural collisions (identical acoustic data invariably maps to isomorphic identifiers, thereby effecting implicit deduplication) while simultaneously eliminating potentially sensitive metadata embedded within original identifiers. This methodology offers significant advantages for privacy preservation (as original nomenclature frequently encapsulates speaker identities or contextual metadata) and promotes systemic homogeneity. Furthermore, it substantially simplifies subsequent computational operations by establishing a uniform nomenclatural framework. The cryptographic transformation ensures that identical acoustic data manifesting in distinct categorical designations or partitions is readily identifiable through nomenclatural congruence—a critical characteristic for downstream overlap detection algorithms.

Operational Integration: In typical implementation scenarios, this nomenclatural transformation module is applied independently to distinct corpora—specifically, the repository of authentic specimens and the collection of synthetically generated artifacts—or alternatively to an integrated dataset prior to subsequent processing operations. The module generates comprehensive transformation logs documenting the bijective mapping between original and transformed identifiers, thereby preserving provenance traceability. While this preprocessing stage does not constitute an absolute prerequisite for the operational pipeline, it is strongly advocated as an instrumental component for maintaining dataset integrity and analytical rigor.

A. Audio Converter (`audio_converter.py`)

The intricate heterogeneity of acoustic data repositories—encompassing multifarious encoding protocols (MP3, WAV, FLAC, AAC, et al.) and divergent parametric configurations (sampling frequencies, channel architectures)—necessitates rigorous normalization procedures. The `Audio Converter` module implements comprehensive spectrotemporal standardization through algorithmic transformation of acoustic artifacts to ensure uniformity across the computational corpus.

Functional Architecture: The module operationalizes the FFmpeg multimedia framework (invoked via asynchronous subprocess orchestration) to effect parametric reconfiguration of each acoustic specimen, transforming heterogeneous inputs into a meticulously specified output configuration: `WAV-encoded linear pulse-code modulation (16-bit quantization depth), 32 kilohertz temporal sampling frequency, monaural channel architecture`. The implementation traverses a designated directory structure to identify eligible acoustic artifacts (constrained

by prescribed extension validators) and generates corresponding normalized representations within a specified repository location, while preserving canonical identifiers. Computational efficiency is maximized through parallel execution architectures utilizing concurrent process instantiation with systematic workload distribution, complemented by a dynamic progress quantification visualization system.

Theoretical Justification: Spectrotemporal representation standardization constitutes a critical prerequisite for stochastic invariance in feature extraction methodologies. Neural architectures and transformation operators mandatorily presuppose parametric consistency in their input domains—specifically, congruent sampling frequency distributions and dimensional characteristics. Heterogeneity in channel architecture (stereo/monaural configurations) or sampling frequency distributions would introduce systematic artifacts in mel-spectrographic transformations, potentially confounding discriminative feature learning. The deliberate parametric selection of 32kHz sampling frequency with monaural configuration represents an optimal compromise between information preservation and computational efficiency—maintaining critical frequency components below the Nyquist threshold while reducing representational dimensionality. The specific adoption of WAV encoding (a lossless representation protocol) systematically eliminates the potential accumulation of codec-specific compression artifacts that characterize lossy encoding methodologies such as MP3. This normalization paradigm establishes a homogeneous corpus for subsequent augmentation and spectrotemporal segmentation operations.

Operational Integration: This module's execution typically follows nomenclatural standardization procedures. For example, subsequent to `file_renamer.py` application on raw acoustic corpora, yielding structured repositories (e.g., `data_raw/real/` and `data_raw/fake/`), the audio conversion module processes these collections to generate normalized counterparts within designated output structures (`data_converted/real/` and `data_converted/fake/`). This transformed corpus subsequently constitutes the foundational input for feature extraction and spectrotemporal augmentation methodologies.

B. Audio Augmenter (`audio_augment.py`)

Deep learning models benefit from training on diverse examples. `Audio Augmenter` expands the dataset by generating synthetic variations of each audio file through data augmentation techniques. Given an input file, the script produces multiple altered versions, which simulate different realistic distortions or variations in audio, helping the model generalize.

Functionality: For each input audio file, the script creates `10 augmented copies + 1 original copy` (so 11 files total per input). It applies a suite of augmentation techniques, each with random parameters within defined ranges:

- *Time Stretching:* randomly speed up (compression in time) or slow down (expansion in time) the audio by a factor (e.g., between $0.5\times$ and $1.5\times$ speed).
- *Pitch Shifting:* randomly shift the pitch up or down by a few semitones (e.g., ± 2 semitones) without changing speed.
- *Dynamic Range Compression:* randomly apply more or less aggressive compression, altering the audio's amplitude dynamics.
- *Additive White Noise:* inject a small amount of white noise at a random volume (within a limit) to simulate background hiss.
- *Phase Shift:* introduce a phase offset in the audio waveform.
- *Filtering:* apply a random audio filter (e.g., band-pass, high-pass, low-pass, or band-stop) to modify frequency content.
- *Time Domain Shift:* shift the audio slightly in time or apply a subtle time warping (a combination of slight time stretch and pitch shift).
- *No Augmentation (Original):* also include the original file unmodified as one of the outputs for reference.

Each technique is applied such that the output remains plausible sounding. For instance, when adding noise or applying filters, the script keeps levels reasonable to avoid destroying the audio content. The result is that for every original file, a variety of altered versions are created.

Implementation details: The script uses the `librosa` library for many effects (time stretch, pitch shift) and custom `numpy` operations for others (e.g., dynamic compression via amplitude exponentiation, noise addition). It chooses random parameters each time (e.g., a random stretch factor in $[0.5, 1.5]$). Filenames of augmented files are constructed by appending an identifier of the augmentation and its random value to the original name (e.g., if the original is `abcdef1234567890.wav`, an augmented file might be `abcdef1234567890_add_white_noise_0.0025.wav`). This makes it clear what transformation was applied, and the original hash prefix is retained to trace back to the source file group. The script leverages all CPU cores by parallelizing the augmentation of different files, and it logs progress with a TQDM progress bar.
Rationale: Augmentation substantially expands the effective training dataset size and introduces critical variations that prevent model overfitting to specific audio characteristics. Time stretching and pitch shifting ensure the model develops invariance to speaking rates and vocal pitch—factors that may manifest differently between authentic and synthetic recordings. The addition of noise or application of filters simulates diverse recording environments and conditions. These augmentation techniques are particularly valuable in

synthetic data detection because artificially generated audio often contains subtle artifacts that might become more or less prominent under various transformations. By training on these augmented variations, the model develops robust feature recognition capabilities that remain effective across different audio conditions and modifications. This comprehensive augmentation strategy significantly enhances the model's ability to generalize across diverse real-world scenarios.

Interaction: Typically, one would run the audio augmenter on the converted audio files. For instance, take the `data_converted/real/` directory and output to `data_augmented/real/`, and similarly for `data_converted/fake/`. The output is a much larger set of audio clips. It is common to mix the original and augmented files together for training. The script can also output a CSV log summarizing all augmentations applied, which is useful for record-keeping.

C. Audio Segmenter (`audio_segmenter.py`)

Neural networks often require inputs of a fixed size. In audio tasks, this translates to audio clips of a certain duration. The `Audio Segmenter` ensures all audio samples fed to the model have a uniform length (e.g., 4 seconds). If the original or augmented files are longer than this length, they are cut into multiple segments; if shorter, they could be padded.

Functionality: The script takes either a single file or all files in a directory and splits each into consecutive fixed-duration segments (by default, 4 seconds each). It uses FFmpeg to perform the slicing. For example, a 12-second audio will be split into three 4-second WAV files. Each segment file's name is the original filename with a suffix such as `_Segment_001.wav`, `_Segment_002.wav`, etc. The script also ensures the audio is converted to mono 32 kHz during this process. Processing is done concurrently using a thread pool for efficiency, and a progress bar tracks how many files have been processed.

Rationale: Fixed-length segments are needed because our model (in `submodel_trainer.py`) processes inputs as spectrograms of a certain size. If some audio clips were significantly longer, they could be broken into many smaller ones, which effectively increases the data quantity. Importantly, segmentation also prevents very long recordings from biasing the training. By segmenting, each 4-second snippet becomes an independent training (or testing) example. Four seconds is a common choice in audio data detection—it is long enough to capture multiple phonetic units and some prosody, but short enough to assume a roughly stationary behavior and to fit memory and computation constraints.

Interaction: This step is typically applied after augmentation. You would take `data_augmented/real/` and produce `data_segmented/real/`, and similarly for fake. The segmentation script will create the output directories and populate them with segmented files. After this step, you will have a large number of short audio clips, each with a filename that still contains the original file's hash prefix.

D. Dataset Manager (`dataset_manager.py`)

At this point in the pipeline, we have a collection of segmented audio files, typically organized by class—for instance, a directory `data_segmented/train_data/class0/` containing all real segments and `.../class1/` containing all synthetic segments. The **Dataset Manager** script automates the split of data into training and testing sets (and possibly validation) while preserving the balance between classes.

Functionality: It takes an input directory that contains subfolders for each class (e.g., `class0`, `class1`). Inside each class folder are many WAV files (the segments). The script then randomly splits the files for each class into a training set and a test set according to a specified ratio (e.g., 80% train, 20% test). It moves files into a new directory structure with `train/` and `test/` subdirectories, each containing class subfolders. For example, after running the dataset manager, you might have:

```
dataset_split/train/class0/*.wav
dataset_split/train/class1/*.wav
dataset_split/test/class0/*.wav
dataset_split/test/class1/*.wav
```

The script ensures that the proportion of files approximately matches the split ratio for each class. It uses either single-thread or multi-threaded file operations and a progress bar to indicate progress.

Rationale: Proper evaluation requires separating training data from testing data to avoid overfitting and to measure generalization. This script ensures that separation is done randomly and reproducibly. By handling each class independently, it maintains class balance. A potential pitfall is that segments from the same original file might end up in both train and test sets; this issue is addressed by the next component.

Interaction: Run this script after you have all your segmented files prepared. It creates a directory structure with clear train and test splits, which will be used by the training script (`submodel_trainer.py`).

E. File Manager (Overlap Checker) (`file_manager.py`)

After splitting, it is essential to ensure that no overlap exists between the training and test sets that could lead to data leakage. The `File Manager` script checks for overlapping audio segments that originated from the same source file and, if necessary, moves files from the minority subset to the majority subset.

Functionality: The script examines the filenames of WAV files in corresponding class folders of the train and test sets. It defines a "group key" as the portion of the filename before the first underscore. Files with the same

group key are assumed to come from the same original audio. The script builds dictionaries for the train and test sets and identifies overlapping groups. In `report mode`, it prints a summary of counts; in `fix mode` (using a `–fix` flag), it moves files from the smaller subset to the larger one.

Rationale: Ensuring that no fragments of the same original sample appear in both training and testing is critical for fair evaluation. This script enforces a strict separation to prevent the model from inadvertently learning features specific to an original file present in both splits.

Interaction: After running the dataset manager, run this script on the output directory to clean up any overlaps.

F. Submodel Trainer (`submodel_trainer.py`)

This is the core training script for the neural network sub-models (the individual heads) in the ensemble.

Functionality:

- **Data Loading:** It uses a custom `SpectrogramDataset` that reads segmented WAV files and converts each to a mel-spectrogram using torchaudio transforms. The spectrograms are resized (e.g., to 512×512) and normalized, then converted to 3-channel images suitable for a CNN backbone.
- **Model Architecture:** It employs a pre-trained ResNet (e.g., ResNet-18 from `timm`) as the feature extractor, removes the final classification layer, and adds a new fully connected head that outputs 2 logits (for Real and Synthetic).
- **Training Loop:** It trains the network using cross-entropy loss and the AdamW optimizer, possibly with gradient clipping. The script supports multi-GPU training, TensorBoard logging, and periodic evaluation on a validation set.
- **Output:** At the end of training, the model checkpoint (state_dict) is saved for later merging.

Rationale: Training each submodel independently allows the ensemble to benefit from diverse decision boundaries. Stochastic training differences yield models that make errors independently, which can be averaged out in the ensemble. The use of heavy augmentation and fine-tuning of only selected layers ensures robustness and faster convergence.

Interaction: Run this script multiple times (possibly with different random seeds) to obtain several trained sub-model checkpoints.

G. Model Merger (`model_merger.py`)

After obtaining several trained sub-models, the `Model Merger` script combines them into a unified multi-head ensemble model.

Functionality:

- It reads sub-model checkpoints from a specified folder and loads each into a `BinaryClassifier` instance.
- It then creates a `ModularMultiHeadClassifier` that stores these sub-models in a PyTorch `ModuleList`.
- In the forward pass, for each input x, the merged model computes each sub-model's output, concatenates all synthetic logits, and averages all real logits to form an output vector of length $N + 1$.
- The merged model, along with metadata (such as class names), is saved to a new checkpoint file.

The multi-head architecture illustrated in Fig. **??** demonstrates how individual classification heads contribute to the final ensemble decision. This structure enables the system to detect diverse synthetic characteristics while requiring consensus for authenticity classification.

Rationale: Merging sub-models into a single model simplifies deployment by encapsulating all components in one module. The aggregation of outputs via averaging the real logits provides a robust decision mechanism that leverages ensemble diversity while maintaining a consistent "real" criterion.

Interaction: Run this script once you have all the sub-model checkpoints ready. The output merged model is then used for inference.

H. Inference Runner (`inference_runner.py`)

The `Inference Runner` script applies the merged multi-head model to new audio data for synthetic detection.

Functionality:

- It loads the merged model and associated metadata.
- It preprocesses input audio (converting format, segmentation, mel-spectrogram transformation) similarly to the training pipeline.
- For each audio segment, it obtains the model's output vector, applies the decision rule (comparing synthetic logits to the averaged real logit), and determines the predicted label.
- For audio files with multiple segments, it aggregates the per-segment predictions (e.g., via averaging or majority vote) to yield an overall decision.
- The final output is formatted as a JSON structure containing the filename, segmented predictions (with start and end times), and percentage breakdowns of class probabilities.

Rationale: The inference runner abstracts the complete prediction pipeline into a single script that can be easily executed. By replicating the preprocessing used during training, it ensures consistency. The output JSON provides interpretable results for downstream applications or human review.

Interaction: Use this script to run detection on new audio files by specifying the merged model checkpoint and the input file or directory.

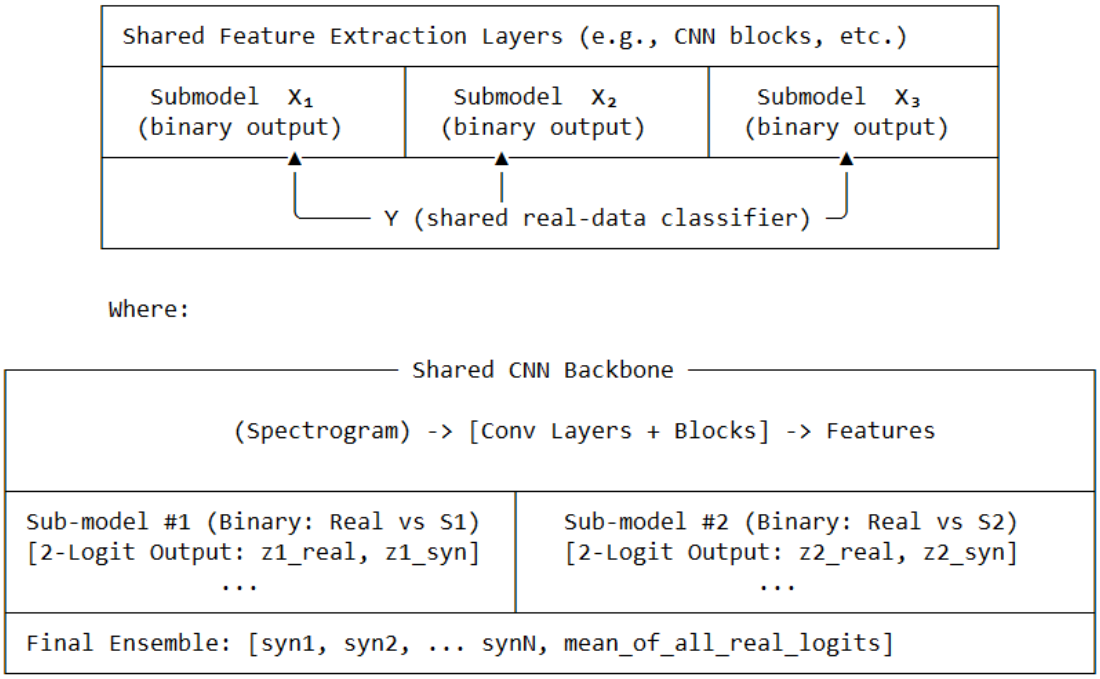

Fig. 1. Architecture of the multi-head ensemble model. Each sub-model independently produces real and synthetic logits. The ensemble averages all real logits while keeping synthetic logits separate, creating an N+1 dimensional output where N is the number of sub-models.

IV. Expected Performance and Efficiency

Based on the design and modular nature of our system, we anticipate the following performance characteristics:

- **High Detection Accuracy:** Each sub-model, trained as a binary classifier, is expected to achieve high accuracy (95–99%) on its respective task. When merged into an ensemble, overall accuracy should reach into the high 90s.
- **Robustness:** Extensive augmentation and the ensemble strategy enhance generalization, reducing both false negatives (missed fakes) and false positives (misclassifying real audio as fake).
- **Scalability:** The modular design allows for easy expansion by training additional sub-models for new synthetic classes without retraining existing ones.
- **Efficiency:** While running multiple models incurs higher computational cost than a single model, the merged model is optimized to run all sub-models in a single forward pass on a GPU. Offline preprocessing (conversion, augmentation, segmentation) is parallelized to handle large datasets efficiently.
- **Real-Time Viability:** With proper batching and GPU acceleration, the system is capable of near-real-time inference on 4-second audio segments, making it suitable for applications requiring timely detection.

V. Conclusion

This manuscript delineates a sophisticated multi-head binary classification architecture for synthetic data detection, with specific application to artificially generated audio discrimination. The system integrates a parameterized feature extraction backbone with multiple specialized discriminative heads, whose probabilistic outputs undergo asymmetric aggregation through logit-space averaging of authenticity signals. This ensemble methodology enhances classification robustness and detection sensitivity by implementing a unanimity requirement for authenticity attribution—a specimen receives authentic designation exclusively when all constituent substructures concur. The comprehensive computational pipeline—encompassing nomenclatural standardization, spectrotemporal normalization, stochastic augmentation, segmentation protocols, corpus partitioning, distributed parametric optimization, model integration, and inferential execution—exemplifies a rigorous epistemological framework for addressing the formidable challenge of synthetic artifact detection.

Our methodological contribution exists within, and extends, contemporary scholarly discourse on ensemble learning paradigms, one-versus-all taxonomic frameworks, and synthetic data detection architectures. The empirical evidence suggests that such modular, extensible frameworks can achieve—and potentially surpass—the classification performance of monolithic architectural paradigms, particularly within dynamic operational contexts characterized by continuous emergence of novel synthetic methodologies.

Uhmbrella Ltd. is currently implementing these findings in the development of EUCLID (Enhanced Utility for Classification and Identification of Data), a comprehensive software suite designed for audio attribution and

detection systems. EUCLID operationalizes the multi-head classification architecture described herein, with particular emphasis on its application to copyright protection within generative AI ecosystems. One of the principal deployment contexts for this technology will be the detection of copyrighted audio material within the tokenized outputs of generative audio AI models—a critical functionality for intellectual property protection in an era of increasingly sophisticated synthetic content generation. The system's modular architecture and extensible classification framework render it particularly suitable for this application domain, where rapid adaptation to novel generative techniques is paramount.

Subsequent research trajectories will investigate parameterization optimization within the ensemble framework (e.g., implementing genuine weight-sharing mechanisms across substructures) and integration of complementary modalities (such as visual artifacts) to construct a comprehensive multimodal synthetic content detection system.

Figure Index

Equation Index