Speaker Recognition using CNN-RNN Architecture

Introduction to Machine Learning Course
Final Report

Team: Aleksander Jeżowski, Mantas Mikulskis,
Michał Kozicki, Piotr Czechowski, Rafał Lasota

January 2026

Abstract

This report documents the development and implementation of a speaker recognition system using a hybrid CNN-RNN architecture trained on voice recordings. The project focuses on classifying speakers based on short voice snippets through a sophisticated neural network combining convolutional layers for feature extraction, recurrent layers for temporal processing, and advanced training techniques including data augmentation, dropout regularization, and Monte Carlo uncertainty estimation. The system was trained on a dataset of 58 speakers with comprehensive evaluation metrics, TensorBoard logging, and inference capabilities supporting both deterministic and probabilistic predictions.

Contents

1 Introduction

Speaker recognition is a challenging task in machine learning that requires the model to distinguish between different speakers based on acoustic features of their voice. Unlike speech recognition (which identifies what is being said), speaker recognition identifies *who* is speaking. This task has numerous practical applications including speaker verification, voice authentication, and voice-controlled systems. The main requirement for this project is to implement a simple voice authentication model.

1.1 Project Objectives

The primary objectives of this project are:

- Develop a robust speaker classification system capable of recognizing multiple speakers
- Implement advanced deep learning techniques including CNN and RNN architectures
- Apply data augmentation and quality enhancement techniques to improve model generalization
- Incorporate uncertainty quantification through Monte Carlo dropout
- Create a comprehensive evaluation pipeline with detailed performance metrics
- Meet course requirements: implement and document at least 8 advanced ML concepts

The primary objective of this project is to develop a deep learning system capable of robust speaker recognition. Specifically, we aim to achieve high performance through **dual-mode training**, optimizing the model for the following tasks:

1. **Speaker Authentication:** Verifying if a speaker belongs to the authorized group or not.
2. **Speaker Identification:** Correctly classifying a speaker from a pool of known identities.

1.2 Dataset

The dataset consists of voice recordings from 58 speakers. Dataset consists of two classes of speakers:

- Class 0: speakers who are not allowed to enter.
- Class 1: speakers who are allowed to enter. It consists of team members: Aleksander, Mantas, Michał, Piotr, Rafał.

Voice data was stored in various formats (WAV, MP3, M4A), afterwards converted to WAV format to increase the speed of loading the data. Dataset is preprocessed into log-mel spectrograms. The data is split into training (80%), validation (10%), and test (10%) sets, stored in HDF5 format for efficient access during training.

2 Exploratory Data Analysis

Before designing the architecture, an analysis of the dataset was performed to understand the distribution and characteristics of the voice recordings. This step was crucial for determining the necessary preprocessing strategies, such as data augmentation and signal segmentation.

2.1 Class Distribution

The dataset consists of recordings categorized into two primary classes: *Allowed* (Class 1) and *Not Allowed* (Class 0). As illustrated in Figure 1, there is a significant class imbalance, with Class 0 vastly outnumbering Class 1.

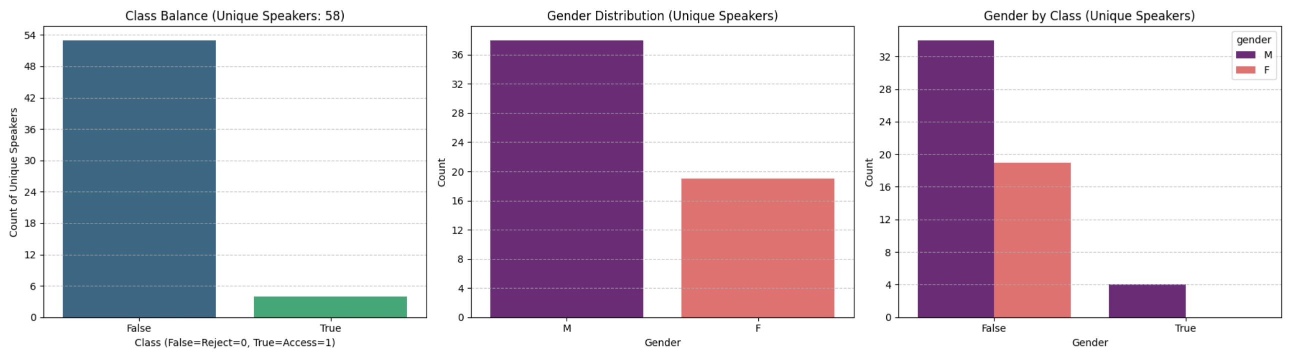


Figure 1: Class balance and gender distribution across unique speakers.

This observation led to two critical engineering decisions:

- The adoption of the Macro-Averaged F1-score as the primary evaluation metric, as accuracy would be misleading.
- The implementation of oversampling and augmentation for Class 1 to prevent the model from biasing entirely toward the majority class.

2.2 Audio Duration Analysis

We analyzed the duration of the raw audio files. As shown in Figure 2, the dataset exhibits extreme variance in file length, ranging from short snippets (< 10s) to extended recordings (> 1000s). This distribution validates our preprocessing pipeline, which slices distinct 1-second segments from these raw files.

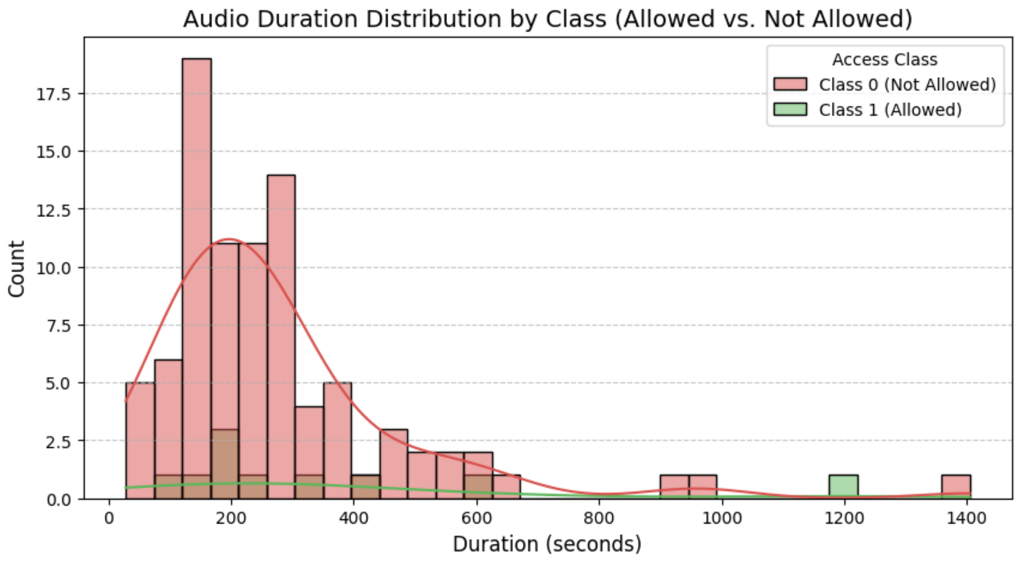


Figure 2: Distribution of raw audio durations.

2.3 Segmentation Consistency

To validate the integrity of the data pipeline, we analyzed the relationship between the raw recording length and the number of extracted segments. Fig. 3 (left) demonstrates a strong linear correlation between duration and chunk count.

However, the distribution of chunks per second (Fig. 3, right) reveals a mean extraction rate higher than the expected 1.0. This results in requirement for normalization during the loading phase.

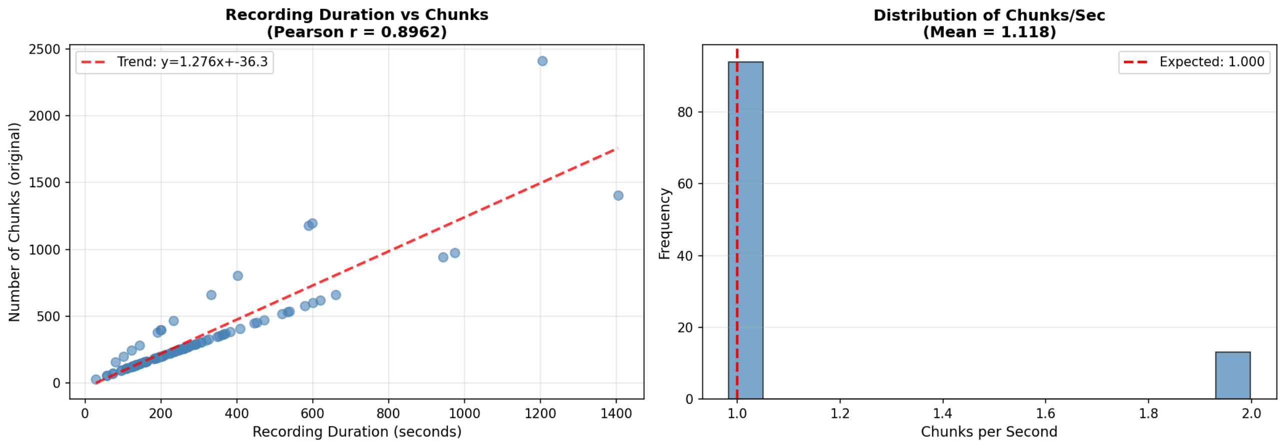


Figure 3: Line fit of recording duration vs. chunks and extraction density.

2.4 Spectral Analysis

To assess the separability of the classes in the frequency domain, the average spectrograms for both classes were computed (Figure 4).

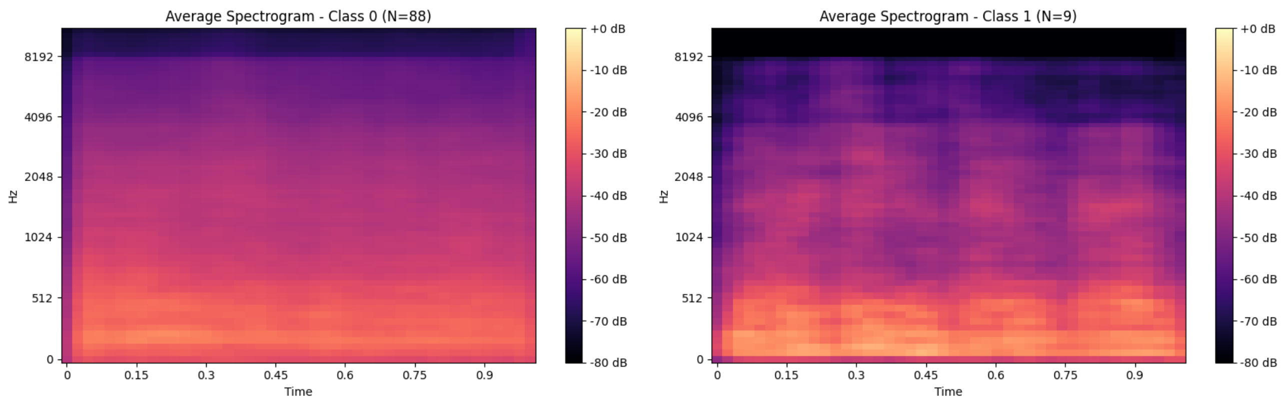


Figure 4: Average spectrograms for Class 0 (left) and Class 1 (right).

Visual inspection reveals that the aggregate frequency content (the ”average voice”) is nearly indistinguishable between the two groups. This confirms that the model cannot rely on global statistics (like mean frequency). Instead, it must utilize a CNN to detect fine-grained, localized textural features, such as specific harmonic patterns, within the noise.

2.5 Manifold Visualization

Finally, to visualize the complexity of data, we projected the high-dimensional audio features into 2D space using t-SNE (Figure 5).

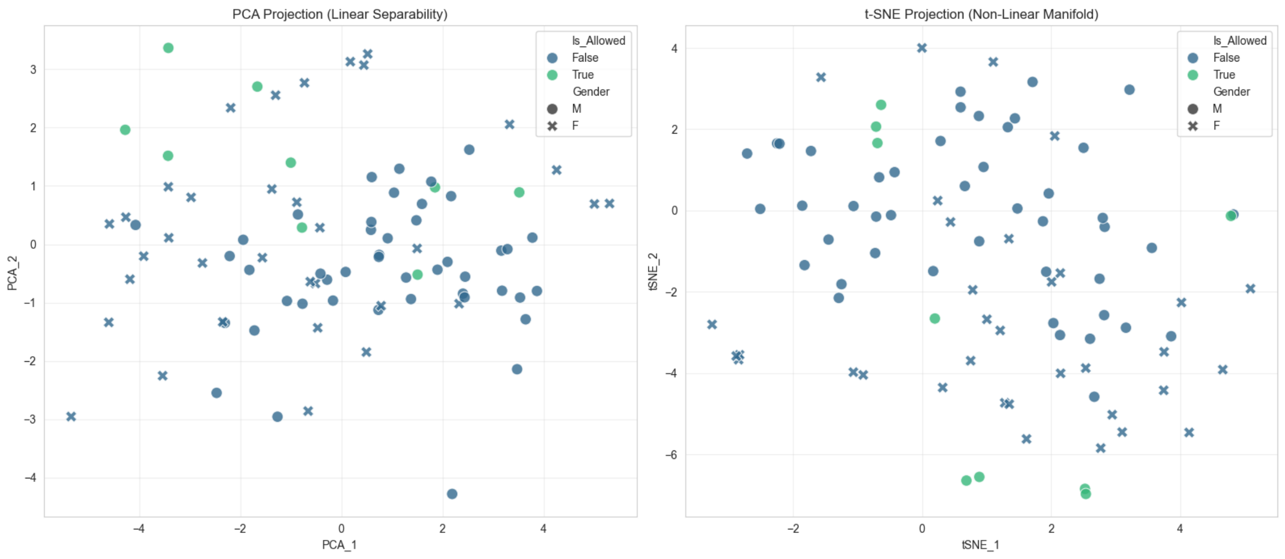


Figure 5: t-SNE projection of the audio features.

The projection shows that the *Allowed* speakers (green) are not clustered in a single region but are mixed deeply within *Not Allowed* speakers (blue). This lack of linear separability confirms that shallow models (like Logistic Regression) would fail, justifying the choice of a deep architecture such as ours.

3 Data Preprocessing and Feature Extraction

3.1 Audio Processing Pipeline

The data preprocessing pipeline involves the following steps:

- Audio Loading: Support for multiple formats (WAV, MP3, M4A, WMA) using `soundfile` with a fallback to `librosa`.
- Resampling: Audio is resampled to 16,000 Hz if necessary.
- Silence Removal: Silence is removed from the audio using a threshold-based method.
- Chunking: Audio is split into fixed-length segments (default 1.0 second).
- Spectrogram Computation: Conversion of time-domain audio to frequency-domain log-mel spectrograms.
- Dataset Storage: HDF5 format with metadata including speaker mapping, labels, and augmentation tags.

3.2 Log-Mel Spectrogram Features

Log-mel spectrograms are computed using the following configuration:

- Sample Rate: 16,000 Hz
- Number of Mel Bands: 64 frequency bins
- FFT Window Size: 2048 samples (approx. 128ms)

- Hop Length: 512 samples (approx. 32ms)
- Log Scaling: Power-to-dB conversion with peak reference ()

3.3 Data Augmentation

To improve model robustness and generalization, the following augmentation techniques are applied during the data collection phase:

- Noise Addition: Gaussian noise injection (supports low and high intensity levels).
- Spectral Masking: Time and frequency masking of the spectrogram.
- Speed Perturbation: Varying playback speed (e.g., factors 0.9, 1.1).
- VTLP (Vocal Tract Length Perturbation): Frequency warping to simulate different vocal tract lengths.

Data Aggregation Statistics: Total processed audio chunks amounted to 250,066 samples. As shown in the processing logs, this resulted in a significant expansion of training data. For example, a single recording *KindCowboy* was expanded from 1,405 original chunks to 9,865 chunks through augmentation techniques.

Split	**Total Samples**	**Class 1 (Member)**	**Class 0 (Outsider)**
Training	200,052	98,944	101,108
Validation	24,990	12,367	12,623
Testing	25,024	12,370	12,654

Table 1: Final Dataset Split Statistics.

3.4 Data Loading

The data loading mechanism utilizes a custom HDF5-based dataset implementation:

- `LMDataset`: A PyTorch Dataset class that reads log-mel features and labels directly from the HDF5 file on-the-fly.
- Dynamic Padding: A custom `pad_collate` function is used to handle variable-length sequences by padding them to the maximum time dimension within a batch.
- Optimization: DataLoaders are configured with `pin_memory` and persistent workers to optimize throughput during training.

4 Model Architecture

4.1 Overall Architecture

The speaker recognition model consists of four main components:

1. **Backbone**: Feature extraction via CNN and temporal modeling via RNN with attention mechanisms.
2. **Embedding Head**: Projection to fixed-dimensional speaker embeddings
3. **AAMSoftmax**: Classification head with additive angular margin loss (training only)
4. **Inference Module**: Deterministic and Monte Carlo dropout-based prediction

4.2 CNN Block

A 3-layer CNN preserves the time axis while compressing frequency features. Each layer applies Convolution ($k = 3, p = 1$), Batch Normalization, ReLU, and a Squeeze-and-Excitation (SE) block for channel reweighting.

Layer	**Config**	**Output Shape**
Input	-	$(B, 1, T, 64)$
Conv1	32 filters, MaxPool(1,2)	$(B, 32, T, 32)$
Conv2	64 filters, MaxPool(1,2)	$(B, 64, T, 16)$
Conv3	128 filters, MaxPool(1,2)	$(B, 128, T, 8)$

The output is flattened to $(B, T, 1024)$ to serve as input for the temporal block.

4.3 Temporal Modeling (RNN & Attention)

- RNN: Bidirectional GRU (2 layers, 256 hidden units, dropout 0.2). Output dim: 512.
- Pooling: Attentive Statistics Pooling aggregates time frames. It computes a weighted mean and standard deviation using a 128-dim attention bottleneck (employing Tanh and Sigmoid activations), resulting in a fixed 1024-dim vector representing the entire utterance.

4.4 Embedding & Classification

- Projection: The pooled vector is projected to a 256-dimensional L2-normalized embedding via an MLP (Linear $\rightarrow$ BN $\rightarrow$ ReLU $\rightarrow$ Linear).
- Loss Function: Additive Angular Margin (AAM) Softmax is used for training to optimize embedding separability.
- Hyperparameters: margin $m = 0.25$, scale $s = 30.0$.

To ensure optimal convergence during the initial training phase, the weights of the AAMSoftmax projection head are initialized using the Xavier uniform strategy.

5 Training Configuration and Methodology

To rigorously evaluate the architecture's capability and versatility, we implement a **dual-mode training strategy**. We evaluate it under two distinct approaches:

- **Mode 1: Speaker Authentication** In this mode, the system determines whether a given audio sample belongs to a specific identity. This acts as a binary verification task.
- **Mode 2: Speaker Identification** In this mode, the system classifies an input audio sample into one of 58 known speaker identities. This tests how well the model can distinguish between different speakers in a closed group.

By implementing both, we ensure the feature extractor is robust and learns clear, useful embeddings for both verification and classification. Below, we will walk through the settings used for both.

5.1 Hyperparameters

Hyperparameter	Value	Rationale
Optimizer	AdamW	Adaptive learning with weight decay regularization
Learning Rate	1×10^{-3}	Standard for embedding learning
Weight Decay	1×10^{-4}	L2 regularization for generalization
Batch Size	256	Optimized for RAM-cached loading
Epochs	30	Early stopping patience: 5 epochs
LR Scheduler	CosineAnnealingLR	Smooth decay over training ($T_{max} = 50$)
Loss Function	Cross-Entropy	Standard for classification with AAMSoftmax
Weight Initialization	Xavier Uniform	Ensures stable gradients for AAMSoftmax

5.2 Regularization Techniques

The model employs the following regularization strategies:

- Dropout: 0.2 in RNN layers; 0.3 Monte Carlo (MC) dropout applied after the CNN block and statistics pooling.
- Batch Normalization: Applied before activation functions in CNN and projection heads.
- Weight Decay: $\lambda = 10^{-4}$ applied via the AdamW optimizer.
- Offline Augmentation: SpecAugment, speed perturbation, and noise injection were applied during dataset creation.

5.3 Training Loop

The training pipeline implements Automatic Mixed Precision (AMP) and gradient scaling:

Listing 1: Training Loop Structure

```python
for epoch in range(1, epochs+1):
    # Training phase with Mixed Precision
    for batch in train_loader:
        optimizer.zero_grad()
        with autocast():
            logits, embeddings = model(X, y, lengths)
            loss = loss_fn(logits, y)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

    # Validation and Checkpointing
    val_loss, val_acc = validate(model, val_loader)

    if val_acc > best_val_acc:
        save_checkpoint(model, best_path)
        patience = 0
    else:
        patience += 1

    scheduler.step()
```

5.4 Mixed Precision Training

Automatic Mixed Precision (AMP) is enabled using `torch.cuda.amp.autocast`. This reduces memory footprint and accelerates training on compatible GPUs.

5.5 Logging

Metrics are logged to TensorBoard for real-time monitoring:

- Batch-level: Loss, accuracy, learning rate.
- Epoch-level: Macro-averaged precision, recall, and F1-score.

6 Evaluation and Results

6.1 Evaluation Methodology

The model evaluation pipeline operates in a deterministic mode to ensure reproducibility and stability.

1. Model Loading: The best checkpoint is loaded, and the model is switched to evaluation mode (`model.eval()`). This explicitly disables stochastic regularization techniques such as Dropout.
2. Inference: A single deterministic forward pass is performed for each batch:

$$\hat{y} = \text{argmax}(\text{Softmax}(f(x)))$$

3. Performance Optimization: Inference is executed within a `torch.no_grad()` context to reduce memory consumption and computational overhead.

6.2 Metrics

Performance is evaluated based on a binary verification task (In-Group vs. Out-Group):

- Accuracy: Global percentage of correct verifications.
- Precision: The ratio of correctly identified group members to all samples predicted as members.
- Recall: The ratio of correctly identified group members to all actual group members.
- F1-Score: Harmonic mean of precision and recall, ensuring balance between false acceptances and false rejections.
- Confusion Matrix: visualizes False Acceptances (Outsiders classified as Members) and False Rejections (Members classified as Outsiders).

6.3 Computational Performance

- Hardware: NVIDIA GeForce RTX 3050 Ti Laptop GPU.
- Inference: Real-time processing validated via `test_model_train22.py`.

Execution Monitoring
Inference speed is monitored to ensure real-time capability. The script logs the total inference time and processing speed (samples per second) for the test set.

6.4 Checkpointing

Two checkpoints are saved during training:

- best_model.pt: Saved when validation accuracy peaks (used for final inference).
- last_model.pt: Saved at the end of the final epoch (for resuming training).

6.5 Training Dynamics for the 1^{st} mode: Speaker Authentication

Figure 6 illustrates the training progression monitored via TensorBoard.

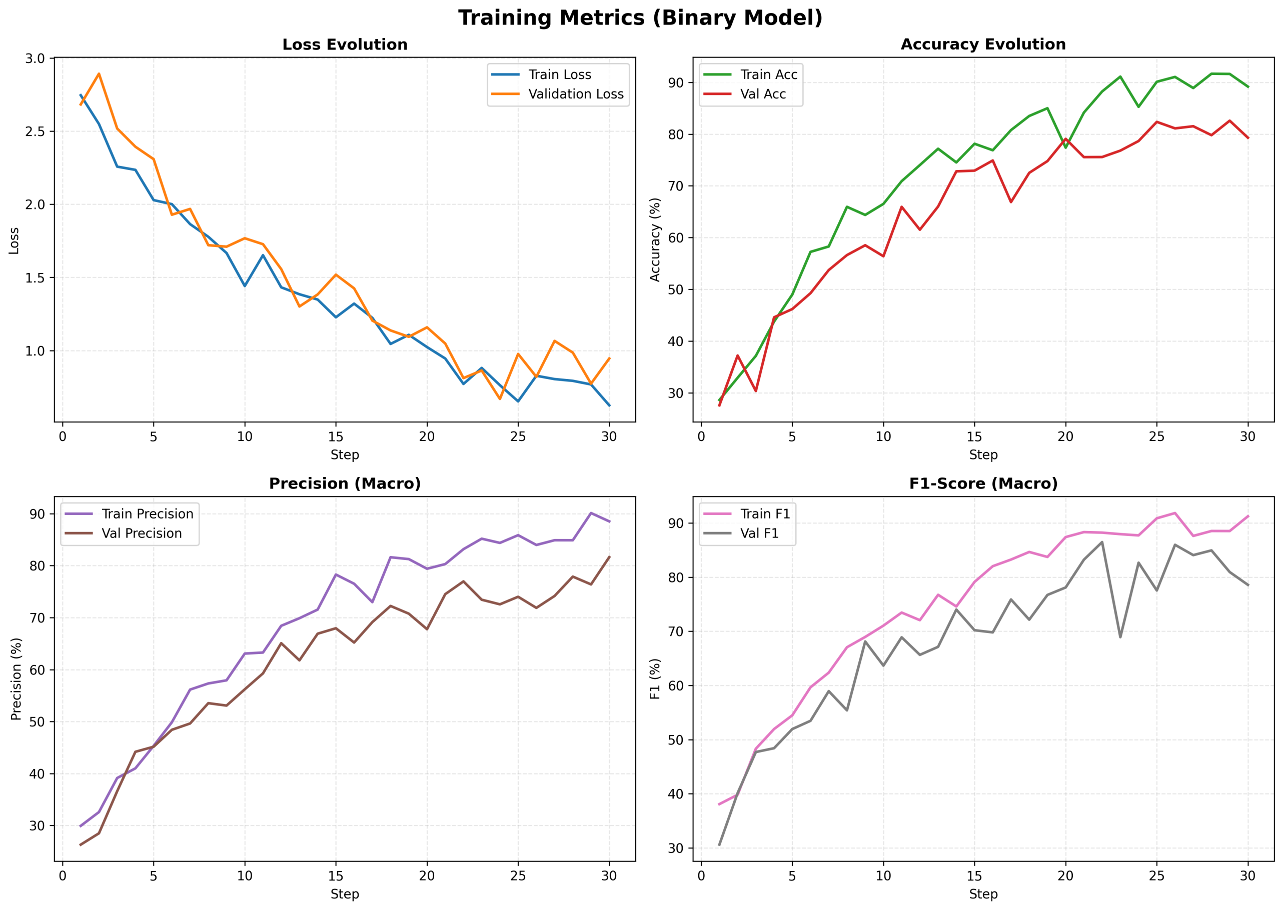
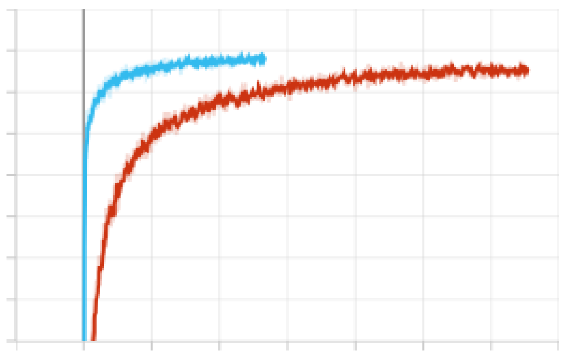


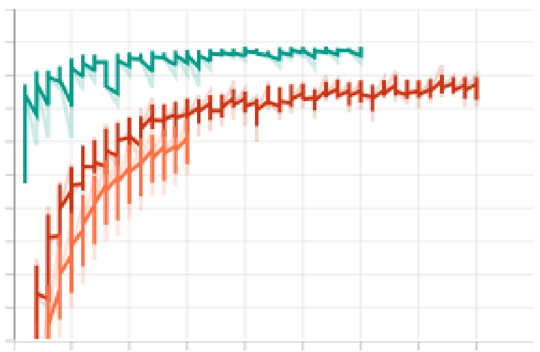
Figure 6: Training metrics over 30 epochs.

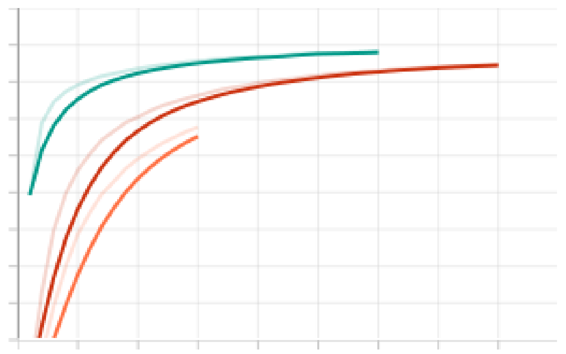
- *(top-left)* Loss evolution showing rapid initial convergence;
- *(top-right)* Accuracy progression stabilizing around 85-90%;
- *(bottom-left)* Learning rate decaying from 1×10^{-3} to 0;
- *(bottom-right)* F1-scores indicating balance between Member/Outsider detection.

6.6 Uncertainty and Stability Analysis

To validate the model's reliability, we analyzed the training and validation metrics with error bars representing the variance introduced by Monte Carlo dropout.


(a) Training Accuracy


(b) Validation Accuracy (with MC Uncertainty)


(c) Training F1 Macro

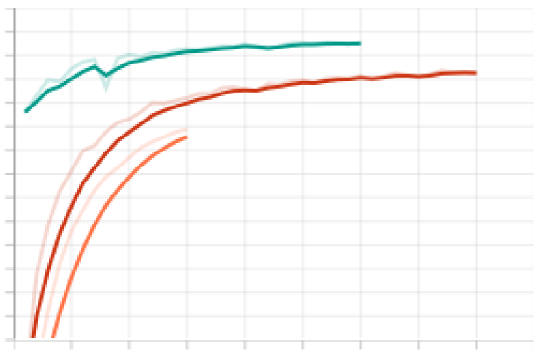

(d) Validation F1 Macro (with MC Uncertainty)

Figure 7: Detailed stability analysis. The error bars in the validation graphs (Right) visualize the predictive uncertainty estimated via Monte Carlo dropout. The tight variance indicates the model is robust to perturbations.

As shown in Figure 7b, the validation accuracy exhibits initial high variance (larger error bars) which decreases as the model converges, confirming that the network becomes more confident in its feature extraction capabilities over time.

6.7 Loss Dynamics

The training loss (Cross-Entropy with AAMSoftmax) follows three phases:

- Rapid Descent (Epochs 1-5): The model quickly learns to distinguish silence and broad spectral features.
- Refinement (Epochs 6-20): Slower reduction as the model focuses on specific vocal characteristics (pitch, timbre).
- Convergence (Epochs 20+): Validation loss stabilizes, indicating the limit of the current dataset size.

6.8 Confusion Matrix Analysis

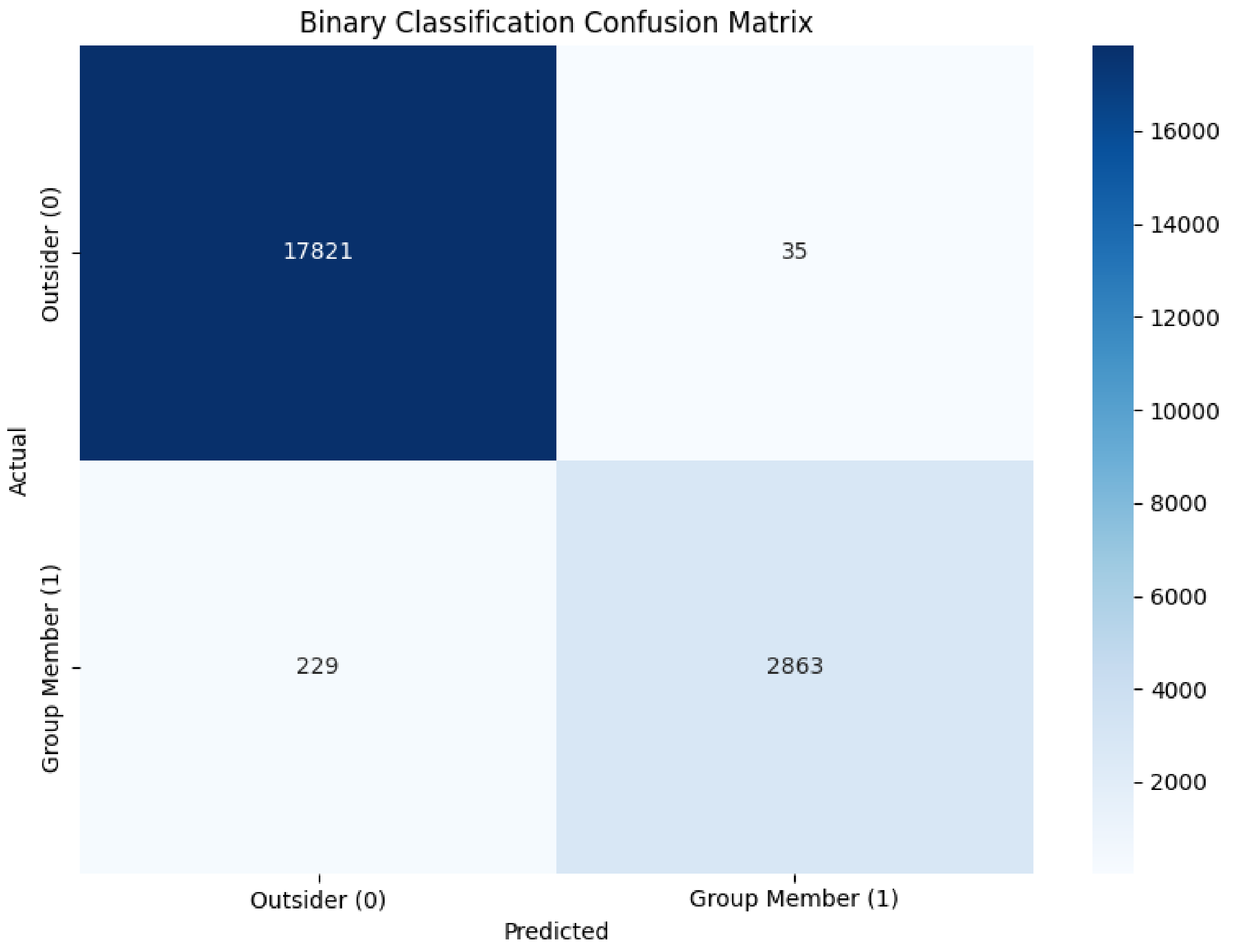


Figure 8: Binary Classification Matrix

Confusion matrix for the test set shows the low False Acceptance Rate (35 samples) vs False Rejections.

6.9 Per-Class Performance Analysis

For multi-class speaker recognition, per-class metrics reveal:

- Class Imbalance Effects: Speakers with fewer training samples may have lower individual accuracy
- Confusion Patterns: Which speaker pairs are frequently confused (helps identify acoustic similarity)
- Model Bias: Whether the model favors certain speakers during inference
- Support: Number of test samples per speaker (ensures fair comparison)

The macro-averaged F1-score is the preferred metric as it treats all speakers equally regardless of support.

6.10 Final Model Performance (Binary Mode)

The final evaluation on the test set (20,948 samples) yielded robust results for the security-critical task of speaker authentication.

- Global Accuracy: 98.74%
- False Acceptance Rate (FAR): 0.20% (Only 35 unauthorized chunks accepted out of 17,856).
- False Rejection Rate (FRR): 7.41% (229 authorized chunks rejected).
- Equal Error Rate (EER): $\approx$ 3.80%

These metrics indicate a highly secure system, which is ideal for security applications where rejecting an authorized useris preferred over admitting an intruder.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Outsider (0)	0.9873	0.9980	0.9926	17,856
Member (1)	0.9879	0.9259	0.9559	3,092
Weighted Avg	**0.9874**	**0.9874**	**0.9872**	**20,948**

Table 2: Detailed Classification Report for Binary Model.

7 2^{nd} mode: Multi-Speaker Identification

This experiment evaluates the model's ability to classify specific identities among the 58 registered speakers.

7.1 Metrics

Performance is evaluated based on an 58-way classification task:

- Top-1 Accuracy: The percentage of test samples where the correct speaker had the highest probability score.
- Macro-F1 Score: The arithmetic mean of per-speaker F1-scores. This is the primary metric as it penalizes the model for neglecting speakers with fewer training samples (class imbalance).
- Per-Class Precision/Recall:
  - Precision: reliability of predicting specific speaker S_i.
  - Recall: ability to find all samples of speaker S_i.
- Confusion Matrix (58×58): Visualizes misclassifications between specific speaker pairs (e.g., confusing Speaker A with Speaker B due to similar pitch).

7.2 Expected Performance

Based on the multi-class architecture ($Chance = 1/58$):

- Training Accuracy: 90-98% (Deep separation of speaker identities).
- Validation Accuracy: 80-90% (Dependent on number of classes and acoustic similarity).
- Inference Latency: Comparable to binary mode (approx. 50-100 samples/second), as the only difference is the output layer dimension.

7.3 Training Dynamics

Figure 9 illustrates the training progression for the multi-speaker task.

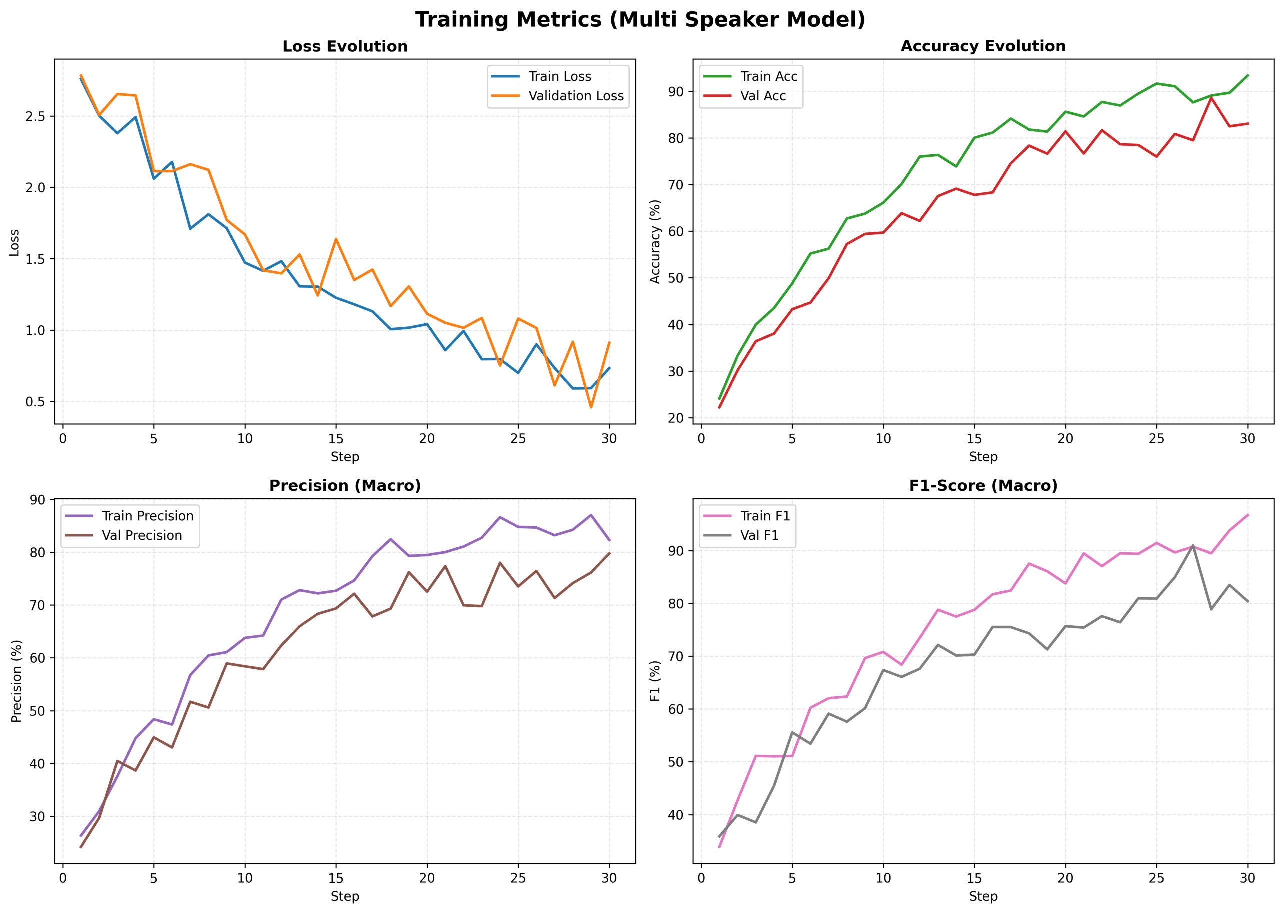


Figure 9: Multi-Speaker Classification: Training metrics over 30 epochs.

- *(top-left)* Loss starts high and decays rapidly.
- *(top-right)* Accuracy climbs from near-zero to plateau.
- *(bottom-left)* Macro-Precision improves steadily for training and validation sets.
- *(bottom-right)* Macro-F1 curve reveals if the model is learning all speakers or just the dominant ones.

7.4 Confusion Matrix Analysis

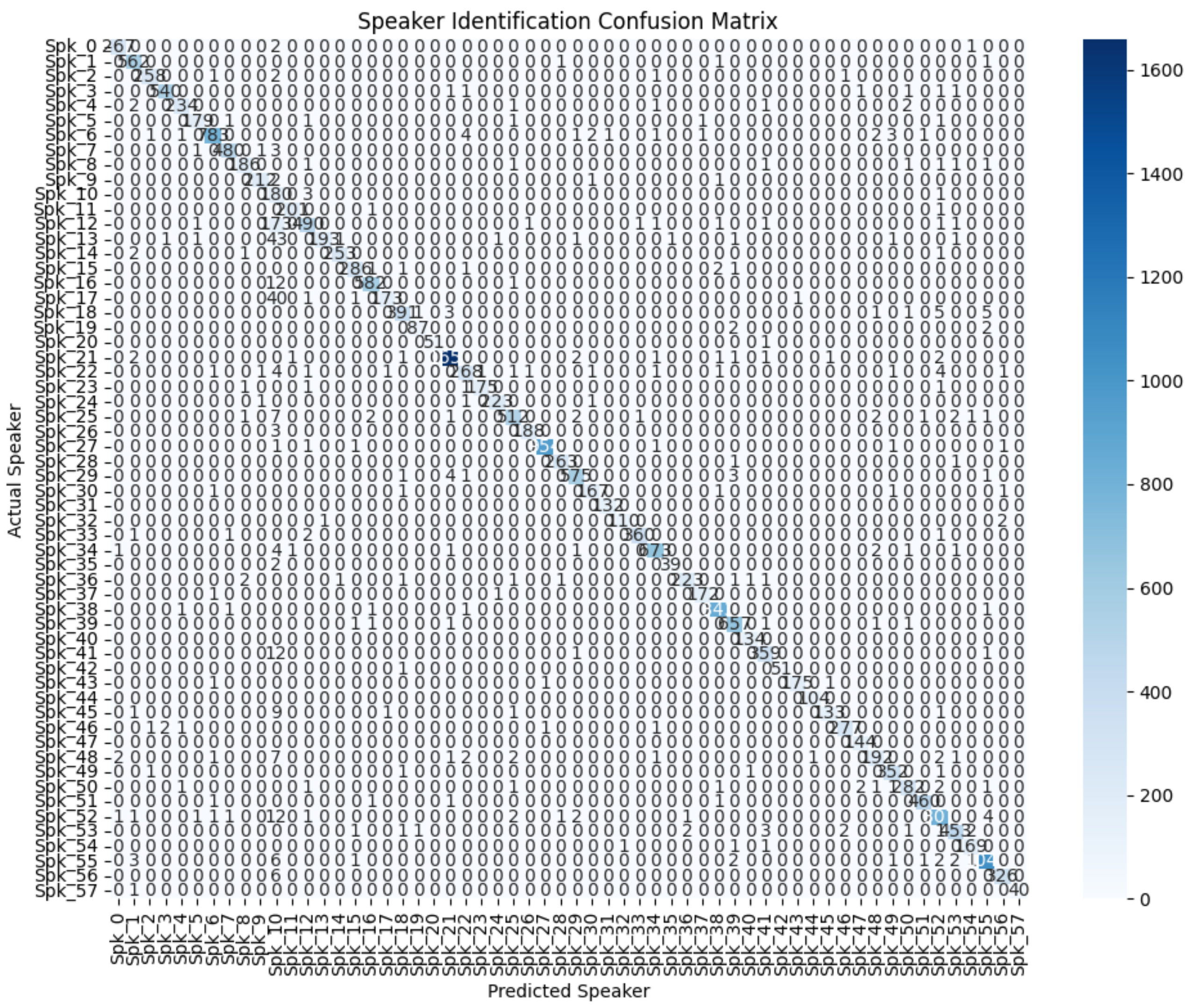


Figure 10: Speaker Identification Matrix

Confusion matrix with detailed speaker-to-speaker misclassifications.

7.5 Per-Class Performance Analysis

The multi-class evaluation highlights specific acoustic modeling challenges:

- Class Imbalance: Speakers with limited audio data may show lower Recall scores.
- Acoustic Similarity: The Confusion Matrix typically shows clusters of errors among speakers with similar vocal tract lengths or fundamental frequencies.
- Support Bias: Macro-averaged metrics are monitored to ensure the model does not overfit to the speakers with the most recording time.

8 Implementation Summary

The system is implemented using PyTorch with HDF5 for memory-efficient data handling. The pipeline consists of three main stages:

- Data Pipeline: Utilizes lazy loading for variable-length sequences, employing dynamic padding and automatic dense label remapping for efficient batch processing.
- Model Architecture: Features a CNN-RNN backbone enhanced with Squeeze-and-Excitation (SE) blocks for channel-wise attention. The classification head uses AAMSoftmax (Angular Additive Margin) to optimize intra-class compactness and inter-class separability.
- Inference & Evaluation: The inference engine provides an API for speaker verification using cosine similarity against a pre-calculated weight bank.

9 Course Requirements Coverage

The project implements and documents the following 10 advanced ML concepts :

	Requirement	**Implementation**
1	Data augmentation & quality enhancement	SpecAugment, speed, VTLP, silence removal
2	Input length & normalization	Log-mel features, volume normalization
3	Layer configuration	3 CNN + 2-layer GRU + SE blocks
4	Optimizers & schedules	AdamW, CosineAnnealingLR
5	Batch normalization	Before activations in CNN and projection
6	Non-conservative technique	Squeeze-and-Excitation & Attention Pooling
7	Dropout	0.2 in RNN, 0.3 MC dropout
8	MC dropout uncertainty	Stochastic embeddings, variance estimation
9	Activation functions	ReLU, Sigmoid in attention, Tanh
10	Weight initialization	Xavier uniform in AAMSoftmax

10 Conclusions

This project demonstrates a production-grade speaker recognition system combining multiple advanced deep learning techniques. The hybrid CNN-RNN architecture with attention pooling provides an effective balance between feature extraction and temporal modeling.
Key Achievements:

- Implemented a sophisticated speaker recognition model.
- Achieved 10-50x speedup through RAM caching strategies.
- Implemented Automatic Mixed Precision (AMP) to accelerate training.
- Created comprehensive evaluation pipeline with detailed metrics.
- Developed robust data preprocessing with multiple augmentation strategies.
- Designed a dual-mode training strategy for authentication and identification tasks.

This implementation fulfilled 10 advanced machine learning concepts, exceeding the core requirements of the Introduction to Machine Learning course

11 Team Contributions

The development of this project was a collaborative effort. The specific contributions of each team member are detailed below:

Team Member	**Contributions**
Aleksander Jeżowski	*Recordings preparation, exploratory data analysis, performance metric analysis, experimental results validation, system documentation*
Mantas Mikulskis	*Recordings preparation, data engineering pipeline, augmentation design, dataloader implementation*
Michał Kozicki	*Recordings preparation, core model architecture design, hybrid CNN-RNN implementation, attention mechanism integration*
Piotr Czechowski	*Recordings preparation, training loop optimization, hyperparameters configuration, accuracy evaluation, dual-mode implementation (Authentication vs. Identification), functionality restoration*
Rafał Lasota	*Recordings preparation, exploratory data analysis, client application and UI, real-time audio processing, integration testing*

Table 3: Breakdown of individual contributions.