Multi-Label Musical Instrument Recognition Using Real and Synthetic Audio Data

Şeyma Betül İSKENDER
Dept. of AI and Data Engineering
Istanbul Technical University
iskenders22@itu.edu.tr

Ahmet Selim ÖZEL
Dept. of AI and Data Engineering
Istanbul Technical University
ozelah23@itu.edu.tr

***Abstract*—This paper presents a comprehensive approach to multi-label musical instrument recognition using a combination of real-world and synthetically generated audio data. We train three deep learning architectures: a Convolutional Neural Network with Channel and Spatial Attention (CBAM-CNN), a Multi-Scale Convolutional Recurrent Neural Network (MS-CRNN), and a Transformer-based model (PaSST). Synthetic polyphonic audio mixtures are generated from isolated NSynth instrument recordings using a structured polyphonic-complexity approach, creating balanced training data across different polyphony levels. The models are evaluated on the OpenMIC 2018 dataset and tested on real-world audio examples. Our results demonstrate that synthetic data augmentation improves model performance, with the PaSST transformer achieving a macro-F1 score of 0.671 at threshold 0.2 when trained on the combined dataset, compared to 0.643 with OpenMIC-only training. An ensemble of all three models further improves robustness and performance.**

***Index Terms*—multi-label classification, musical instrument recognition, synthetic data generation, transformer models, audio tagging**

I. Introduction

Musical instrument recognition in polyphonic audio is a challenging task in music information retrieval (MIR). Unlike single-label classification, real-world audio often contains multiple instruments playing simultaneously, requiring models to predict multiple binary outputs for each instrument class. This multi-label classification problem is further complicated by the scarcity of large-scale, accurately labeled polyphonic datasets.

The OpenMIC 2018 dataset [1] provides real-world multi-label annotations, but with limited size (20,000 samples) and potential class imbalance. To address these limitations, we propose generating synthetic polyphonic audio mixtures from isolated instrument recordings in the NSynth dataset [2], creating controlled training examples with known ground-truth labels.

This work makes several contributions: (1) a structured methodology for generating synthetic polyphonic audio with balanced polyphonic complexity, (2) comprehensive evaluation of three diverse architectures (CNN, CRNN, Transformer) on the multi-label task, (3) analysis of synthetic data augmentation impact through controlled experiments, and (4) real-world validation on audio examples.

II. Related Work

A. Multi-Label Audio Classification

Multi-label audio classification has been addressed using various deep learning architectures. Kong et al. [4] introduced PANNs, large-scale pretrained audio neural networks that demonstrate strong performance on audio tagging tasks. Gong et al. [3] proposed PaSST, an efficient transformer architecture for audio classification that achieves state-of-the-art results on AudioSet.

B. Synthetic Data Generation

Synthetic data generation for audio has been explored in various contexts. Chen et al. [6] investigated musical instrument recognition using synthetic mixtures, demonstrating the value of controlled data generation. Our approach extends this by implementing a structured polyphonic-complexity framework that ensures balanced representation across different polyphony levels.

C. Ensemble Methods

Ensemble methods have shown effectiveness in polyphonic audio tagging [7]. Combining diverse architectures (Transformer, CNN, CRNN) leverages complementary strengths: transformers excel at long-range dependencies, CNNs capture local patterns, and CRNNs model temporal sequences.

III. Methodology

A. Dataset Description

1) OpenMIC 2018: OpenMIC 2018 [1] contains 20,000 ten-second audio clips with multi-label annotations for 20 instrument classes: accordion, banjo, bass, cello, clarinet, cymbals, drums, flute, guitar, mallet_percussion, mandolin, organ, piano, saxophone, synthesizer, trombone, trumpet, ukulele, violin, and voice. Each sample may contain multiple instruments, with binary labels indicating presence.

2) NSynth Dataset: NSynth [2] provides 305,979 isolated instrument notes across 11 instrument families: bass, brass, flute, guitar, keyboard, mallet, organ, reed, string, vocal, and synth_lead. Each note is 4 seconds long at 16 kHz, with metadata including instrument family, pitch, and velocity.

3) IRMAS Dataset: IRMAS (Instrument Recognition in Musical Audio Signals) [8] is a single-label instrument recognition dataset containing isolated instrument recordings. Each audio clip is labeled with a single dominant instrument class. IRMAS is used exclusively for pretraining the CBAM-CNN and MS-CRNN models to help them learn fundamental timbre characteristics before fine-tuning on the multi-label OpenMIC task. The pretraining stage allows these models to develop robust feature representations for individual instruments, which are then adapted for multi-label classification.

B. Synthetic Data Generation

We implement a structured polyphonic-complexity approach to generate synthetic multi-label audio mixtures from NSynth isolated recordings. The methodology ensures balanced representation across different polyphony levels while maintaining accurate label supervision.

1) Polyphonic Complexity Framework: The generation process creates mixtures with polyphony levels $k \in \{0, 1, 2, \ldots, 11\}$, where k represents the number of simultaneously active instruments. For each polyphony level k, we generate $N = 200$ mixtures, resulting in 2,200 total synthetic samples.

2) Instrument Family Mapping: We establish a mapping from NSynth instrument families to OpenMIC tags:

- **Supervised families**: bass $\rightarrow$ bass, flute $\rightarrow$ flute, guitar $\rightarrow$ guitar, keyboard $\rightarrow$ piano, mallet $\rightarrow$ mallet_percussion, organ $\rightarrow$ organ, vocal $\rightarrow$ voice, synth_lead $\rightarrow$ synthesizer
- **Unsupervised families**: brass, reed, string (allowed in audio but not supervised in labels)

The constraint ensures that for $k \geq 1$, at least one supervised family is included, preventing mixtures with only unsupervised families at low polyphony levels.

3) Generation Pipeline: The synthetic data generation pipeline consists of the following steps:

1) **Family Selection**: For polyphony level k, randomly select k instrument families, ensuring at least one supervised family when $k \geq 1$.
2) **Note Selection**: For each selected family, randomly choose an isolated note from the NSynth pool for that family.
3) **Audio Processing**:
    - Load and convert to mono if stereo
    - Resample to target sample rate (16 kHz)
    - Loop or trim to target duration (10 seconds)
    - Peak normalize to prevent clipping
    - Apply random gain: $G \sim \text{Uniform}(-10, 0)$ dB
4) **Mixing**: Sum all processed waveforms element-wise to create the polyphonic mixture:

$$x_{\text{mix}}[n] = \sum_{i=1}^{k} G_i \cdot x_i[n] \tag{1}$$

where $x_i[n]$ is the i-th normalized waveform and G_i is its gain.

5) **Final Normalization**: Peak normalize the mixed audio to prevent clipping.
6) **Label Generation**: Create binary label vector $\mathbf{y} \in \{0, 1\}^{20}$ and mask vector $\mathbf{m} \in \{0, 1\}^{20}$:

$$y_j = \begin{cases} 1 & \text{if instrument } j \text{ is present in the mixture} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$m_j = \begin{cases} 1 & \text{if instrument } j \text{ is supervised (included in loss)} \\ 0 & \text{if instrument } j \text{ is unsupervised (excluded from loss)} \end{cases} \tag{3}$$

The mask vector $\mathbf{m}$ plays a crucial role in the training process. When $m_j = 1$, the instrument j is **supervised**, meaning its prediction contributes to the loss computation during training. When $m_j = 0$, the instrument is **unsupervised**, and its prediction is excluded from the loss calculation. This masking mechanism prevents the model from learning incorrect associations for instruments that are not directly represented in the NSynth families used for synthetic generation. In our synthetic dataset, only 8 out of 20 OpenMIC tags are supervised (bass, flute, guitar, mallet_percussion, organ, piano, synthesizer, voice), while the remaining 12 tags are unsupervised (mask=0) to prevent false supervision from ambiguous or unavailable instrument families.

4) Data Statistics: The generated synthetic dataset contains:

- Total samples: 2,200
- Polyphony distribution: 200 samples per level ($k = 0$ to $k = 11$)
- Audio format: 16 kHz, 10 seconds, mono, 16-bit PCM
- Supervised tags: 8 (bass, flute, guitar, mallet_percussion, organ, piano, synthesizer, voice)
- Unsupervised tags: 12 (all others)

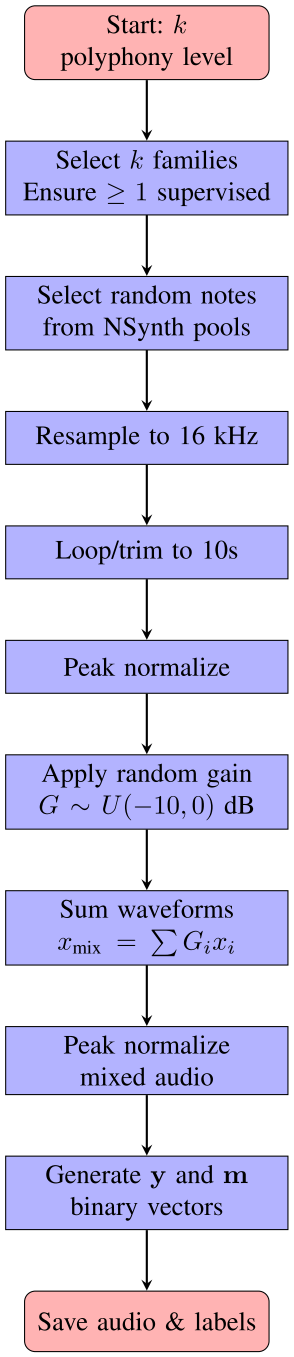
C. Model Architectures

1) CBAM-CNN: The Convolutional Block Attention Module (CBAM) CNN [5] incorporates both channel and spatial attention mechanisms:

- **Input**: Log-mel spectrogram
- **Backbone**: ResNet-like CNN with CBAM attention modules
- **Attention**: Sequential channel attention followed by spatial attention
- **Pretraining**: Pretrained on IRMAS dataset for single-label instrument recognition
- **Fine-tuning**: Classification head replaced for 20-class multi-label output

2) Multi-Scale CRNN: The Multi-Scale Convolutional Recurrent Neural Network combines multi-scale feature extraction with temporal modeling:

- **Input**: Log-mel spectrogram
- **Multi-Scale CNN**: Parallel convolutional branches with different kernel sizes
- **Feature Fusion**: Concatenation of multi-scale features



Start: k polyphony level
Select k families Ensure ≥ 1 supervised
Select random notes from NSynth pools
Resample to 16 kHz
Loop/trim to 10s
Peak normalize
Apply random gain $G \sim U(-10, 0)$ dB
Sum waveforms $x_{\text{mix}} = \sum G_i x_i$
Peak normalize mixed audio
Generate $\mathbf{y}$ and $\mathbf{m}$ binary vectors
Save audio & labels

Fig. 1. Synthetic Data Generation Pipeline

- **BiLSTM**: Bidirectional LSTM for temporal sequence modeling
- **Pretraining**: Pretrained on IRMAS dataset
- **Classification**: Fully connected layer for 20-class output

3) PaSST Transformer: PaSST (Patch-based Spectrogram Transformer) [3] is a vision transformer adapted for audio classification. The architecture processes log-mel spectrograms as image patches:

- **Input**: Log-mel spectrogram (128 mel bins $\times$ time frames)
- **Patch Embedding**: Convolutional patch embedding (16×16 patches, stride 10)
- **Transformer Backbone**: 12 transformer blocks with 768 hidden dimensions
- **Classification Head**: Linear layer mapping to 20 output logits
- **Pretraining**: Pretrained on AudioSet, fine-tuned on our task

We use the `passt_s_swa_p16_128_ap476` checkpoint, which achieves strong performance on AudioSet.

D. Training Procedure

1) Pretraining Stage: Before training on the multi-label task, the CBAM-CNN and MS-CRNN models undergo a pretraining stage using the IRMAS dataset. This pretraining serves to initialize the feature extractors with instrument-specific timbre knowledge.

Pretraining Process:

- **Dataset**: IRMAS single-label instrument recognition dataset
- **Task**: Single-label classification (one instrument per sample)
- **Objective**: Learn fundamental timbre characteristics and feature representations for individual instruments
- **Training**: Models are trained to classify the dominant instrument in each IRMAS sample
- **Transfer**: After pretraining, the final classification layers are removed, and the learned feature extractor weights are used to initialize the multi-label training stage

This pretraining approach allows CBAM-CNN and MS-CRNN to develop robust feature representations before adapting to the more complex multi-label task. The PaSST transformer, in contrast, uses weights pretrained on AudioSet, a large-scale audio tagging dataset.

2) Data Preprocessing: All audio is preprocessed consistently:

- Convert to mono (average channels if stereo)
- Resample to target sample rate (32 kHz for PaSST, 16 kHz for CNN/CRNN)
- Pad or trim to 10 seconds duration
- Compute log-mel spectrogram (128 mel bins)

3) Training Configuration: All models are trained with:

- **Loss Function**: Binary Cross-Entropy with Logits (BCEWithLogitsLoss)
- **Masked Loss**: Loss computed only on supervised labels (masked BCE)
- **Optimizer**: AdamW with learning rate 10^{-4} and weight decay 10^{-2}
- **Batch Size**: 32
- **Mixed Precision**: Automatic Mixed Precision (AMP) enabled
- **Early Stopping**: Patience of 6 epochs based on validation loss
- **Maximum Epochs**: 20

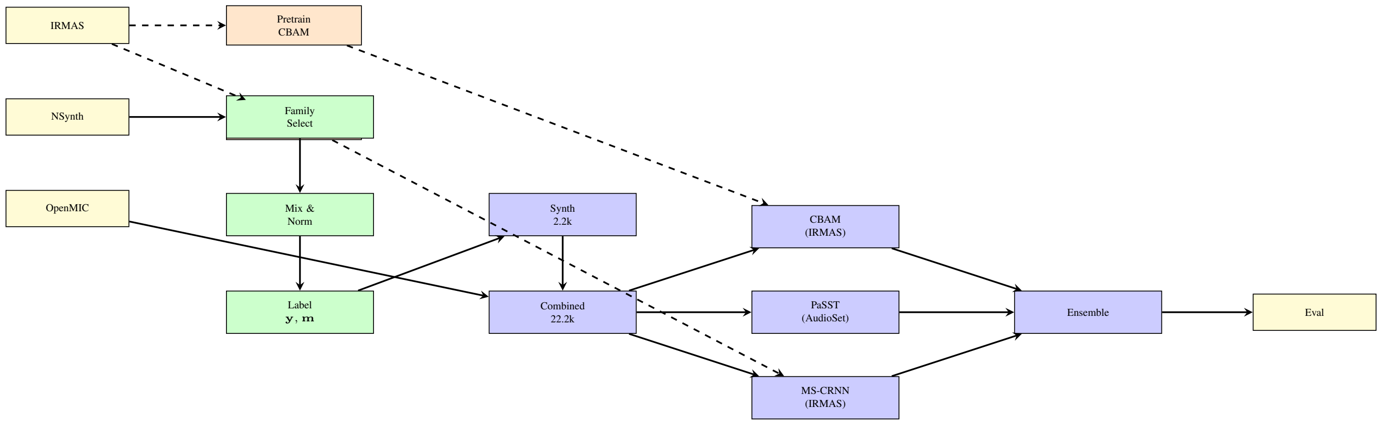


Fig. 2. System Architecture and Training Pipeline. Dashed arrows indicate pretraining stage, solid arrows indicate main training flow.

The masked loss function is defined as:

$$\mathcal{L} = \frac{1}{\sum_j m_j} \sum_{j=1}^{20} m_j \cdot \text{BCE}(\hat{y}_j, y_j) \tag{4}$$

where m_j is the mask value for instrument j. When $m_j = 1$, the instrument's prediction contributes to the loss computation; when $m_j = 0$, it is excluded. This ensures that only supervised tags (those with reliable labels) contribute to the training objective, preventing the model from learning incorrect patterns from unsupervised or ambiguous labels.

4) Data Splits: The combined dataset (OpenMIC + Synthetic) is split as:

- Training: 17,760 samples (80%)
- Validation: 2,220 samples (10%)
- Test: 2,220 samples (10%)

For the OpenMIC-only baseline, the split is:

- Training: 16,000 samples (80%)
- Validation: 2,000 samples (10%)
- Test: 2,000 samples (10%)

E. Ensemble Method

We construct an ensemble by averaging the sigmoid outputs (probabilities) from all three models:

$$p_{\text{ensemble}} = \frac{1}{3} \sum_{m \in \{\text{PaSST}, \text{CBAM}, \text{MS-CRNN}\}} \sigma(\text{logits}_m) \tag{5}$$

where σ is the sigmoid function. The ensemble prediction is thresholded at 0.5 for binary classification.

IV. Experiments

A. Experimental Setup

All experiments are conducted on NVIDIA High-Ram A100 GPUs using Google Colab. Training time per epoch ranges from 5.5 to 6.5 minutes depending on the model architecture. We perform two main experiments:

1) **OpenMIC + Synthetic**: Training on combined dataset (22,200 samples)
2) **OpenMIC Only**: Training on OpenMIC dataset only (20,000 samples) for comparison

B. Evaluation Metrics

We evaluate performance using:

- **Macro-F1 Score**: Average F1 score across all 20 instrument classes
- **Per-Class F1**: Individual F1 score for each instrument
- **Precision and Recall**: Per-class precision and recall
- **Test Loss**: Binary cross-entropy loss on test set

Two probability thresholds are evaluated:

- **Threshold 0.2**: More permissive, increases recall
- **Threshold 0.5**: Stricter, increases precision

C. Real-World Testing

We test the best-performing model (PaSST trained on OpenMIC + Synthetic) on five real-world audio examples downloaded from YouTube:

1) Violin, Cello, Piano trio
2) Guitar, Clarinet, Piano
3) Trumpet, Saxophone, Trombone
4) Saxophone, Piano
5) Voice, Guitar

Each audio is preprocessed identically to training data and evaluated at multiple thresholds to assess detection accuracy.

V. Results

A. Model Performance Comparison

Table I shows the test set performance of all three models trained on the combined OpenMIC + Synthetic dataset.

TABLE I
Model Performance on Test Set (OpenMIC + Synthetic)

Model	**Test Loss**	**Macro-F1@0.2**	**Macro-F1@0.5**
PaSST	0.0916	0.6707	0.6163
CBAM-CNN	0.8252	0.1286	0.1869
MS-CRNN	0.6561	0.2934	0.4062
Ensemble	–	0.2788	0.4770

TABLE II
IMPACT OF SYNTHETIC DATA ON PASST PERFORMANCE

Dataset	Test Loss	Macro-F1@0.2	Macro-F1@0.5
OpenMIC Only	0.0749	0.6427	0.5744
OpenMIC + Synthetic	0.0916	0.6707	0.6163
Difference	+0.0167	+0.0280	+0.0419

B. Synthetic Data Impact Analysis

Table II compares PaSST performance with and without synthetic data augmentation.

The results demonstrate that synthetic data augmentation improves F1 scores (particularly at threshold 0.5, with +4.19% improvement) despite a slight increase in test loss. This suggests that synthetic data helps the model learn better classification boundaries, improving recall for underrepresented classes.

C. Per-Class Performance

Table III shows per-class F1 scores for PaSST (OpenMIC + Synthetic) at threshold 0.5, highlighting the top and bottom performing classes.

TABLE III
PER-CLASS F1 SCORES (PASST, OPENMIC + SYNTHETIC, THRESHOLD 0.5)

Instrument	F1	Precision	Recall
Top 5 Classes			
piano	0.811	0.786	0.838
synthesizer	0.782	0.772	0.792
violin	0.773	0.836	0.719
organ	0.772	0.772	0.772
drums	0.754	0.812	0.704
Bottom 5 Classes			
clarinet	0.042	1.000	0.021
trombone	0.260	0.867	0.153
banjo	0.368	0.941	0.229
cello	0.492	0.879	0.341
trumpet	0.517	0.787	0.385

Classes with high precision but low recall (e.g., clarinet, trombone) indicate conservative predictions, suggesting potential for threshold calibration or class-specific thresholds.

D. Real-World Testing Results

Table IV summarizes real-world testing results on audio examples downloaded from YouTube.

The model demonstrates strong performance on some instrument combinations (violin detection at 98.6%) but struggles with others (voice detection at 2.2%). This variability suggests domain adaptation challenges when moving from training data to real-world audio. Figure 3 provides a detailed visualization of the detection results for one example audio clip, showing probability distributions, detection status at different thresholds, and the audio waveform.

TABLE IV
REAL-WORLD TESTING RESULTS (THRESHOLD 0.2)

Audio File	Expected	Detected (prob.)
violin_cello_piano	violin, cello, piano	vln (0.986), pno (0.401), vlc (0.228)
guitar_clarinet_piano	guitar, clarinet, piano	clt (0.370), gtr (0.027), pno (0.007)
trumpet_sax_trombone	trumpet, sax, trombone	sax (0.511), tpt (0.190), tbn (0.098)
saxophone_piano	saxophone, piano	sax (0.897), pno (0.014)
voice_guitar	voice, guitar	gtr (0.643), vce (0.022)

E. Ablation Study: Synthetic Data Contribution

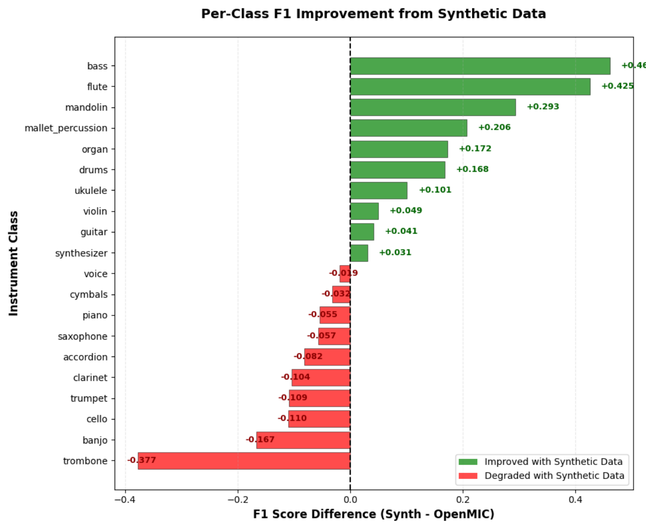
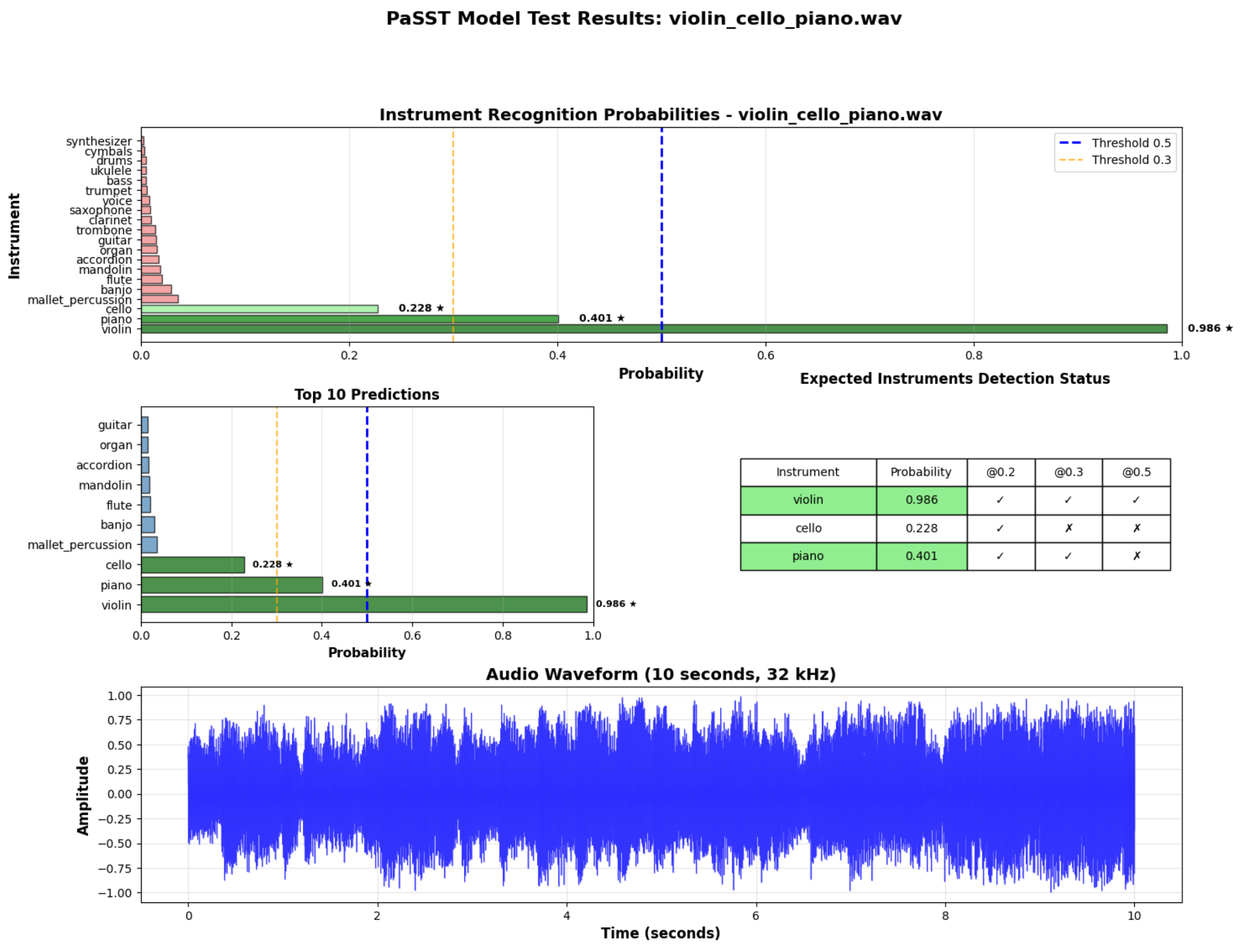


Fig. 4. Per-class F1 score improvement from synthetic data augmentation. Positive values (green) indicate improvement, negative values (red) indicate degradation.

Figure 4 visualizes per-class F1 improvement from synthetic data augmentation. Classes like bass, flute, and mandolin show significant improvements (+46.1%, +42.5%, +29.3% respectively), while some classes (trombone, banjo) show degradation, indicating class-specific effects of synthetic data.



Instrument	Probability	@0.2	@0.3	@0.5
violin	0.986	✓	✓	✓
cello	0.228	✓	✗	✗
piano	0.401	✓	✓	✗

Fig. 3. Real-world testing result of an example audio clip. Shows detection status for a test audio example at threshold 0.2.

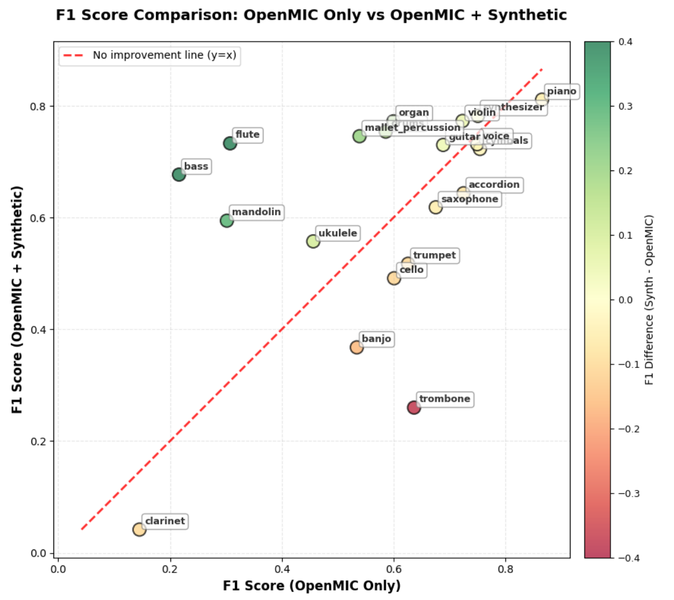


Fig. 5. F1 score comparison: OpenMIC-only vs OpenMIC+Synthetic. Points above the diagonal (y=x) indicate improvement with synthetic data.

Figure 5 shows the F1 score comparison between OpenMIC-only and OpenMIC+Synthetic models, with points above the diagonal indicating improvement.

VI. Discussion

A. Synthetic Data Effectiveness

The structured polyphonic-complexity approach successfully generates diverse training examples. The improvement in macro-F1 scores (+2.8% at threshold 0.2, +4.2% at threshold 0.5) demonstrates the value of synthetic augmentation, particularly for classes with limited training examples.

However, the class-specific impact varies significantly. Some instruments benefit substantially (bass, flute), while others show minimal or negative impact (trombone, banjo). This suggests that the NSynth-to-OpenMIC mapping may not perfectly capture all instrument characteristics, particularly for instruments not directly represented in NSynth families.

B. Limitations and Future Work

Several limitations are identified:

- **Supervision Gap**: Only 8 of 20 OpenMIC tags are supervised in synthetic data, limiting augmentation benefits for unsupervised classes
- **Domain Mismatch**: Real-world YouTube audio may differ from training distribution (compression, reverb, mixing)
- **Class Imbalance**: Some instruments (clarinet, trombone) remain challenging despite augmentation.

Future work could explore:

- Expanding NSynth family coverage to include more OpenMIC instruments
- Domain adaptation techniques for real-world audio
- Per-class threshold optimization
- Advanced mixing strategies (spatialization, effects)

VII. Conclusion

This work presents a comprehensive approach to multi-label musical instrument recognition using real and synthetic audio data. The structured polyphonic-complexity framework for synthetic data generation provides balanced, diverse training examples that improve model performance. The PaSST transformer, trained on the combined dataset, achieves a macro-F1 score of 0.671 at threshold 0.2, with synthetic data contributing a +2.8% improvement. Real-world testing on YouTube audio demonstrates the model's practical applicability while highlighting domain adaptation challenges.

The contributions include: (1) a reproducible methodology for generating synthetic polyphonic audio with controlled complexity, (2) comprehensive evaluation of three diverse architectures, (3) quantitative analysis of synthetic data impact, and (4) real-world validation. These results provide a foundation for future work in multi-label audio classification and synthetic data generation.

References

[1] J. Kim, J. Urbano, C. Liem, and A. Hanjalic, "OpenMIC: An Open Dataset for Multiple Instrument Recognition," in *Proc. ISMIR*, 2018.

[2] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," in *Proc. ICML*, 2017.

[3] K. Koutini, H. Eghbal-zadeh, M. Dorfer, and G. Widmer, "PaSST: Efficient Transformer for Audio Classification," in *Proc. ICASSP*, 2022.

[4] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, "PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 28, pp. 2880–2894, 2020.

[5] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional Block Attention Module," in *Proc. ECCV*, 2018.

[6] K. Chen, W. Zhang, S. Dubnov, G. Xia, and W. Li, "The Effect of Data Augmentation and Network Depth on Musical Instrument Classification," in *Proc. ISMIR*, 2017.

[7] Y. Wu, K. Chen, and Y. Zhang, "Ensemble Methods for Polyphonic Audio Tagging," in *Proc. ICASSP*, 2020.

[8] J. J. Bosch, J. Janer, F. Fuhrmann, and P. Herrera, "A Comparison of Sound Segregation Techniques for Predominant Instrument Recognition in Musical Audio Signals," in *Proc. ISMIR*, 2012.