Available online at website: https://jurnal.iaii.or.id/index.php/RESTI

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 10 No. 5 (2026) 1340 - 1348 e-ISSN: 2580-0760

PATSE: PyTorch-Native Polar Encoding for Runtime Stability in Audio Processing

Nuk Ghurroh Setyoningrum[1], Ema Utami[2], Kusrini[3], Ferry Wahyu Wibowo[4]
[1,2,3,4]Department of Informatics, Universitas Amikom Yogyakarta, Yogyakarta, Indonesia

[*]*nuke@students.amikom.ac.id*

Abstract

Efficient and stable audio feature extraction is essential for real-time AI systems, yet conventional spectral representations often incur preprocessing overhead and framework-conversion dependencies. This study proposes PATSE (Polarized Amplitude Time Spiral Encoding), a PyTorch-native spatial–temporal encoding framework that converts one-dimensional audio into CNN-compatible two-dimensional polar representations while preserving temporal–amplitude information. Evaluated on 100 infant cry samples, PATSE achieved competitive CPU runtime (5.37 ms), improved latency stability over Mel-Spectrogram, and 91% classification accuracy with a Macro-F1 score of 0.85 using a lightweight CNN. These results suggest that PATSE provides an effective balance between computational efficiency, framework integration, and discriminative representation in the evaluated CPU-based infant cry setting, while broader validation is reserved for future work.

Keywords: audio feature extraction; deep learning; infant cry classification; polarized amplitude time spiral encoding; real-time inference

How to Cite: N. G. Setyoningrum, E. Utami, K. Kusrini, and F. W. Wibowo, "PATSE: PyTorch-Native Polar Encoding for Runtime Stability in Audio Processing", *J. RESTI (Rekayasa Sist. Teknol. Inf.)* , vol. 10, no. 5, pp. 1340–1348, Oct. 2026, doi: 10.29207/resti.v10i5.7776.
*Permalink/*DOI: https://doi.org/10.29207/resti.v10i5.7776

Received: May 1, 2026
Accepted: September 14, 2026
Available Online: October 7, 2026

This is an open-access article under the CC BY 4.0 License
Published by Ikatan Ahli Informatika Indonesia

1. Introduction

Infant cry classification is an important topic in computational paralinguistics with applications in pediatric healthcare, caregiver support, and early diagnosis, where accurate recognition of cry types enables timely intervention and reduces uncertainty [1-3]. Although deep learning has improved classification performance, its effectiveness depends heavily on audio feature representation [4-6]. Most approaches rely on STFT-based features such as mel-spectrograms, MFCCs, and spectrograms due to their compatibility with CNNs [4-7]. However, these methods have limitations in preserving complete signal characteristics, reducing effectiveness in modern deep learning and real-time applications [8].

In addition, librosa-based preprocessing may introduce computational overhead from PyTorch–NumPy conversions and operates outside the native PyTorch computational graph. Frequency-domain magnitude representations capture temporal information indirectly and may discard Fourier phase information, potentially reducing temporal detail relevant to infant cry signals [9-11]. Furthermore, conventional representations provide limited visual interpretability, as they fail to capture amplitude time interactions within a unified spatial structure, reducing their usefulness for analysis and debugging [12], [13]. This limitation is critical in clinical contexts where interpretability is essential [14], [15]. Therefore, this study primarily focuses on computational stability while also examining downstream classification performance.

Recent advances in audio representation have been driven by pretrained models such as AST[16], wav2vec 2.0 , HuBERT [17], and BEATs , which leverage large-scale pretraining to learn contextual audio representations. In contrast, conventional representations such as spectrograms, Mel-Spectrograms, and MFCCs provide computationally simpler front-end representations but may introduce preprocessing and framework-conversion overhead when implemented outside the native deep-learning framework. Within this context, PATSE is investigated as a PyTorch-native spatial–temporal front-end representation emphasizing computational efficiency, runtime stability, framework integration, and suitability for integration into real-time AI pipelines. The present

study does not experimentally compare PATSE with pretrained transformer-based audio models; these models are discussed only to establish the broader methodological context [18-20] .

The primary contributions of this work can be summarized as follows, (1) The previously introduced PATSE representation [12] is extended into a PyTorch-native spatial–temporal framework that transforms one-dimensional audio signals into two-dimensional polar spiral representations encoding temporal progression and amplitude within a unified geometric structure. (2) A fully PyTorch-native implementation is introduced, eliminating NumPy–PyTorch conversion overhead and enabling seamless integration with GPU-compatible PyTorch pipelines. (3) A runtime stability evaluation framework combining an experimental Runtime-SNR indicator with CoV, latency risk, and statistical testing is investigated. (4) Runtime benchmarking across multiple infant cry samples and feature-extraction methods is conducted under standardized input conditions. (5) The study analyzes runtime efficiency, classification performance, and computational characteristics relevant to real-time AI pipelines.

While PATSE was previously introduced for infant cry classification and data augmentation [12] , the present study does not claim the original PATSE representation itself as a new contribution. Instead, this work extends the previous study by implementing PATSE as a PyTorch-native processing pipeline and systematically evaluating its runtime efficiency, execution stability, and system-level behavior. The present contribution therefore lies primarily in the runtime-oriented implementation and evaluation framework, including runtime dispersion analysis using Runtime-SNR and CoV, latency characterization, and statistical comparison with conventional audio feature-extraction methods.

2. Methods

2.1. PATSE: Polarized Amplitude Time Spiral Encoding

PATSE shifts audio representation from frequency-domain decomposition to spatial coordinate-based encoding by mapping temporal progression into a spiral in polar space, where amplitude is represented by radial distance and time by angular position. This produces a CNN-compatible image while preserving essential temporal–amplitude information often lost in conventional frequency-domain representations[12].

Figure 1 provides a conceptual overview of the PATSE transformation process, the equations displayed in the schematic are simplified for visualization, while the formal mathematical formulation implemented in the PyTorch-native pipeline is presented in Equations 1–6.

The PATSE pipeline comprises four sequential stages implemented using native PyTorch operations within then.Module framework, enabling direct integration with PyTorch-based learning pipelines and GPU-compatible execution.

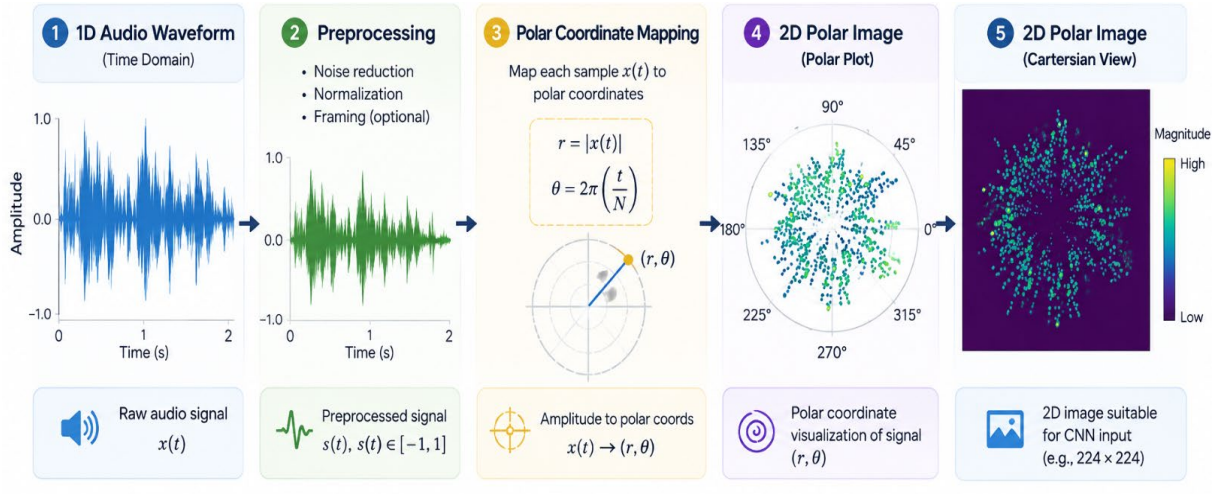


Figure 1. Conceptual illustration of the PATSE architecture and transformation pipeline from a 1D audio waveform to a 2D polar-coordinate image representation. The mathematical expressions shown in the figure are schematic and are intended to illustrate the processing concept, the formal batch-aware formulation used in this study is provided in Equations (1)–(6).

The core PATSE transformation expressed in Equations (1)–(6) is adapted from the original PATSE formulation reported in [12]. In the present study, the formulation is reformulated using batch-aware notation and explicit temporal normalization to support a fully PyTorch-native implementation. Therefore, these equations represent an implementation-oriented extension of the previously published PATSE formulation rather than a newly introduced encoding principle.

Given an input audio waveform $x \in R^{B\times T}$, amplitude normalization is performed independently for each sample as shown in Equations 1 and 2.

$$x_{b,t} = \frac{x_{b,t} - x_b min}{x_b max - x_b min + \epsilon} \quad (1)$$

$$x_b min = \min_{0 \le t \ge T-1} x_{b,t}, \qquad x_b max = \max_{0 \le t \ge T-1} x_{b,t} \quad (2)$$

and ϵ=10^{-6} prevents division by zero.

The temporal index is normalized as shown as Equation 3.

$$T_t = \frac{t}{T-1}, \qquad t = 0, \dots, T-1 \quad (3)$$

The angular coordinate is defined as Equation 4.

$$\theta_t = 2\pi f_{baseTt} \quad (4)$$

f_{base} controls the number of rotations. The radial coordinate is presented by Equation 5.

$$r_{b,t} = x_{b,t} \quad (5)$$

The Cartesian coordinates are then obtained as Equation 6.

$$u_{b,t} = r_{b,t} \cos(\theta_t), \qquad v_{b,t} = r_{b,t} \sin(\theta_t) \quad (6)$$

Unlike Equations 1–6, which are adapted from the original PATSE formulation [12] , Equations 7–10 explicitly define the pixel-coordinate mapping, boundary handling, and scatter-max rendering procedure used in the PyTorch-native implementation developed in the present study. For an output image of size $H \times W$, the Cartesian coordinates are mapped to pixel indices as shown in Equations 7 and 8.

$$p_{b,t}^x = round\left[\frac{u_{b,t}+1}{2}(W-1)\right], \quad (7)$$

$$p_{b,t}^y = round\left[\frac{v_{b,t}+1}{2}(H-1)\right] \quad (8)$$

The indices are clipped to their valid ranges as shown in Equation 9.

$$p_{b,t}^x \in [0, W-1], \quad p_{b,t}^y \in [0, H-1] \quad (9)$$

Scatter max rendering is defined as Equation 10.

$$I_b(i,j) = \max_{\{t: p_{b,t}^x = j, p_{b,t}^y = i\}} x_{b,t} \quad (10)$$

If no sample is assigned to pixel (i, j) then $I_b(i,j) = 0$.

The one-dimensional Gaussian kernel is defined as Equation 11.

$$g(k) = \frac{exp\left(-\frac{k^2}{2\sigma^2}\right)}{\sum_{q=-K}^{K} exp\left(-\frac{q^2}{2\sigma^2}\right)}, k = -K, \dots K \quad (11)$$

The separable implementation reduces the computational complexity from $O(HWk^2)$ to $O(HWk)$.

PATSE is evaluated as an integrated pipeline whose sequential processing stages produce a CNN-compatible representation; therefore, the reported performance reflects the complete pipeline rather than individual components.

2.2. Baseline Methods

Three baseline feature extraction methods were chosen for comparison, representing the range of commonly employed approaches in audio classification research.

The Mel-spectrogram is generated using librosa.feature.mel spectrogram with a 22050 Hz sampling rate, 1024-point FFT, 512 hop length, and 128 Mel bands, then converted to decibel scale via librosa.power_to_db. The Mel filter bank applies perceptual frequency warping that approximates human auditory sensitivity, making it a widely used representation for CNN-based audio classification[21].

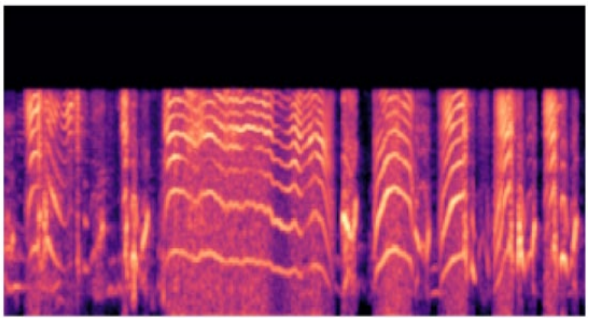


Figure 2. Mel-Spectrogram of Infant Cry Signal

Figure 2 Mel-spectrogram representation of an infant cry signal, illustrating the time–frequency energy distribution where color intensity reflects amplitude variations across temporal frames.

Mel-Frequency Cepstral Coefficients (MFCCs) are computed using librosa.feature.mfcc with 13 coefficients, a 22050 Hz sampling rate, 1024-point FFT, and 512 hop length. As the DCT of the log-Mel spectrum, they provide a compact representation of the spectral envelope. While computationally efficient, their low-dimensional structure limits compatibility with CNNs that require square-like input formats [7].

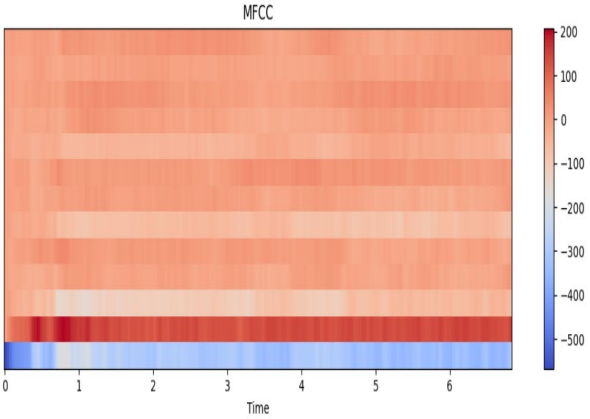


Figure 3. MFCC Representation of Infant Cry Signal

Figure 3 shows the MFCC representation of a baby's cry, where time is mapped horizontally, cepstral coefficients vertically, and color intensity represents DCT-transformed spectral energy. Although compact and efficient, MFCC captures limited temporal detail and is less suitable for CNN-based image processing.

The linear spectrogram is computed using librosa.stft (1024-point FFT, 512 hop length) and converted to a decibel-scale magnitude. Unlike Mel-spectrograms, it preserves full frequency resolution, providing a more detailed but higher-dimensional representation.

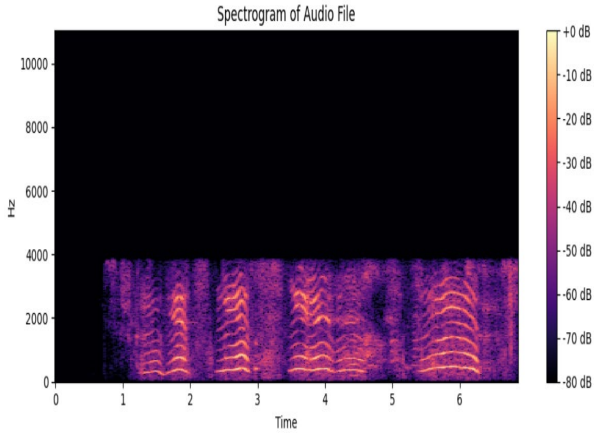


Figure 4. Linear Spectrogram of Infant Cry Signal

Figure 4 shows a linear spectrogram of infant cry, where time and frequency are represented on the axes and color intensity indicates energy in decibels, preserving full frequency resolution with higher dimensionality.

2.3 Positioning Relative to Learned Audio Representation Models

Modern pretrained audio representation models, including the Audio Spectrogram Transformer (AST)[16], wav2vec 2.0, HuBERT[17], and BEATs, provide powerful learned representations through large-scale pretraining and contextual feature modeling.

These models are conceptually different from PATSE, which is designed as a lightweight PyTorch-native front-end representation rather than a large-scale pretrained audio model. Therefore, AST, wav2vec 2.0, HuBERT, and BEATs were not included as experimental baselines in the present runtime benchmark. They are discussed only to position PATSE within the broader landscape of contemporary audio representation methods. The empirical comparisons in this study are restricted to Mel-Spectrogram, MFCC, and linear Spectrogram under the same CPU-based experimental conditions. Direct comparisons with pretrained audio models are reserved for future work.

2.4 Key Architectural Differences

Table 1 compares PATSE with conventional methods (Mel-Spectrogram, MFCC, and Spectrogram) across key aspects including representation domain, computational framework, GPU compatibility, CNN suitability, and NumPy conversion requirements.

Table 1. Architectural comparison of feature extraction methods

Property	PATSE	Mel-Spectrogram	MFCC	Spectrogram
Domain	Polar spatial	Frequency (Mel)	Cepstral	Frequency (linear)
Input → Output	(B,T) → (B,1,H,W)	(B,T) → (B,1,128,F)	(B,T) → (B,1,13,F)	(B,T) → (B,1,513,F)
Framework	PyTorch native	librosa + NumPy	librosa + NumPy	librosa + NumPy
GPU-native	Yes	No (CPU conv.)	No (CPU conv.)	No (CPU conv.)
CNN compatible	Direct (256×256)	Direct (128×F)	Limited (13×F)	Direct (513×F)
NumPy conversion	None	Required	Required	Required

3. Results

3.1. Result

The experiments utilized the publicly available Donate-a-Cry corpus, comprising infant cry recordings categorized into hunger, belly pain, burping, discomfort, and tiredness conditions [4], [12]. To improve robustness and reduce input-specific bias, the runtime evaluation was extended from a single sample to multiple audio samples collected across all classes. A total of 100 randomly selected audio segments were used under identical preprocessing conditions. All audio recordings were resampled to 22050 Hz and standardized to 22050 samples (1 second) through truncation or zero-padding to ensure consistent input dimensions.

Table 2 summarizes the available configuration of the CPU-based benchmark. All methods were evaluated using batch size 1 and a fixed input length of 22,050 samples, with 40 timed iterations following five warm-up iterations. PATSE was executed within the PyTorch pipeline, whereas the conventional baselines required librosa/NumPy processing. The exact CPU model, core/thread configuration, available memory, operating system, and exact software versions were not retained in the original benchmark record and therefore cannot be reported retrospectively without risking inaccurate reconstruction. Consequently, the reported runtime values should be interpreted as measurements obtained under the original controlled CPU benchmark environment rather than as hardware-independent performance estimates.

Table 2. Experimental configuration.

Parameter	Configuration
Compute Device	CPU (no GPU acceleration)
Batch Size	1
Input Length	22050 (1 second at 22050 Hz)
Benchmark Iterations	40
Warm-up Iterations	5
Timing Scope	Feature-extraction execution
PATSE Framework	PyTorch-native
Baseline Framework	librosa/NumPy
NumPy Conversion	Required for Mel/MFCC/Spectrogram
Hardware/Software Details	Not retained in the original benchmark record

Runtime SNR, which is distinct from the conventional signal-to-noise ratio used in audio quality assessment, is introduced here as a logarithmic indicator of relative runtime dispersion as shown in Equation 12.

$$SNR_r = 20 \log_{10} \left(\frac{\mu_r}{\sigma_r + \epsilon} \right), \quad (12)$$

μ_r and σ_r denote the mean and standard deviation of repeated runtime measurements, respectively, and ϵ is a small constant used to prevent numerical instability.

Because the coefficient of variation is defined as Equation 13.

$$CoV = \frac{\sigma_r}{\mu_r} \quad (13)$$

Runtime SNR can be expressed approximately as Equation 14.

$$SNR_r = -20\log_{10}(CoV), \quad (14)$$

Runtime SNR should therefore be interpreted as a logarithmic representation of normalized runtime dispersion rather than as an independent stability construct. Higher values indicate lower runtime variability relative to the mean.

Runtime comparisons were summarized using descriptive statistics obtained from 40 repeated CPU benchmark runs, including mean runtime, standard deviation, best-case runtime, worst-case runtime, and coefficient of variation. The original benchmark record did not retain the complete paired trial-level runtime observations required to reconstruct exact paired inferential statistics. Therefore, exact t statistics, confidence intervals, paired-sample effect sizes, and associated p-values are not reported. The statistical interpretation in this study is consequently restricted to descriptive comparisons under the evaluated benchmark conditions.

Table 3 and Figure 5 show that PATSE achieves competitive runtime efficiency and more controlled latency than Mel-Spectrogram, while Spectrogram and MFCC remain the fastest and most stable methods, respectively.

Table 3. Runtime benchmark results (ms) across 40 runs on CPU

Method	Avg (ms)	Best (ms)	Worst (ms)	Std Dev (ms)
PATSE	5.37	4.478	10.302	1.427
MelSpectrogram	6.245	5.01	14.457	2.109
MFCC	5.515	5.038	8.136	0.667
Spectrogram	2.56	2.253	4.11	0.426

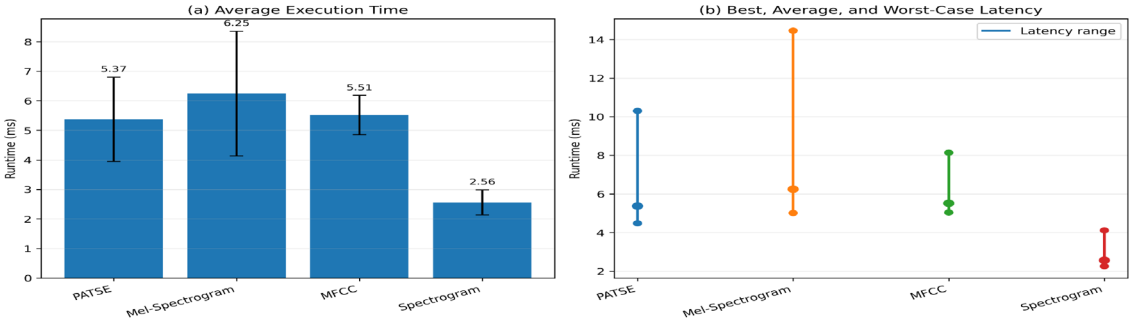


Figure 5. Runtime performance comparison: (a) average execution time with standard deviation error bars; (b) latency range showing best-case, average, and worst-case execution times

As shown in Table 4, the stability analysis reveals that PATSE occupies an intermediate position feature extraction methods. Although MFCC exhibits the lowest runtime variability, its limited spatial structure reduce compability with image-based deep learning models. Conversely, MelSpectogram demonstrates higher variability due to frequency-domain processing overhead. PATSE occupies an intermediate position in runtime stability while providing PyTorch-native framework integration.

Table 4. Runtime stability metrics across all methods

Method	Runtime SNR Best (dB)	Runtime SNR Worst (dB)	CoV (%)	Runtime SNR Gap (dB)
PATSE	15.59	0.74	26.6	14.85
MelSpectrogram	14.08	-2.38	33.8	16.46
MFCC	21.27	6.46	12.1	14.81
Spectrogram	18.44	4.36	16.6	14.08

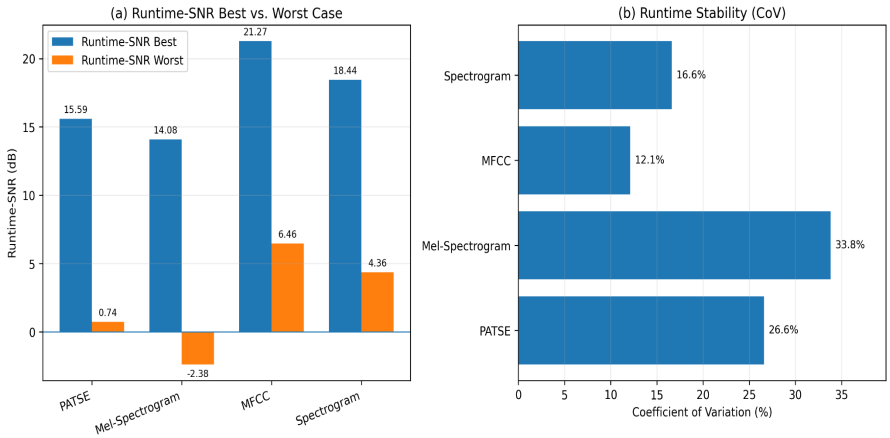


Figure 6. Runtime stability analysis: (a) SNR Best Case vs. Worst Case comparison; (b) Coefficient of Variation showing normalized runtime dispersion

Figure 6 demonstrates that PATSE achieves more controlled runtime behavior than MelSpectrogram, while MFCC and Spectrogram exhibit the highest stability.

Across 40 CPU benchmark runs, PATSE achieved a lower mean runtime than Mel-Spectrogram (5.370 ms vs. 6.245 ms), corresponding to an observed mean difference of −0.875 ms (PATSE − Mel-Spectrogram). The corresponding runtime standard deviations were 1.427 ms and 2.109 ms, respectively. These descriptive results indicate lower average runtime and lower runtime dispersion for PATSE than for Mel-Spectrogram under the evaluated benchmark conditions. However, because the retained benchmark record does not contain the paired trial-level observations, exact paired inferential statistics, including the t statistic, confidence interval, effect size, and p-value, cannot be reliably reconstructed. Accordingly, no inferential significance claim is made from the retained summary statistics.

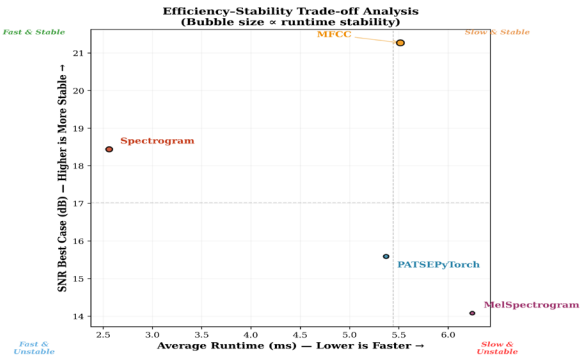


Figure 7. Efficiency–stability trade-off analysis.

Figure 7 illustrates the efficiency–stability trade-off among the evaluated methods. Spectrogram achieves the fastest runtime (2.56 ms), while MFCC shows the lowest runtime variability (CoV = 12.1%). PATSE achieves a runtime of 5.37 ms with a CoV of 26.6%, outperforming Mel-Spectrogram (6.25 ms, CoV = 33.8%) in both measures. Thus, PATSE does not provide the best raw speed or stability, but offers a competitive trade-off while maintaining PyTorch-native integration without NumPy–PyTorch conversion.

Figure 8 shows that Mel-Spectrogram has the highest Worst/Best latency ratio (2.89×), while Spectrogram (1.82×) and MFCC (1.61×) exhibit lower ratios. PATSE falls between these methods (2.30× Worst/Best; 1.92× Worst/Avg), indicating intermediate latency variability under the evaluated CPU benchmark conditions.

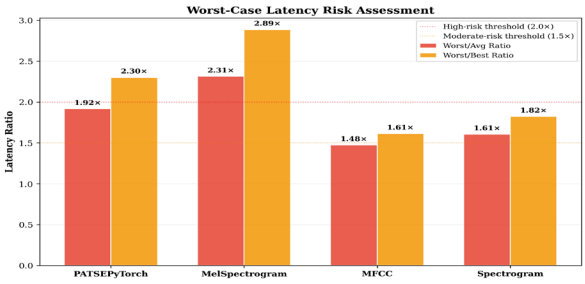


Figure 8. Worst-case latency ratio assessment across the evaluated feature-extraction methods

Figure 9 illustrates the multidimensional characteristics of PATSE and the conventional feature-extraction methods across the evaluated criteria.

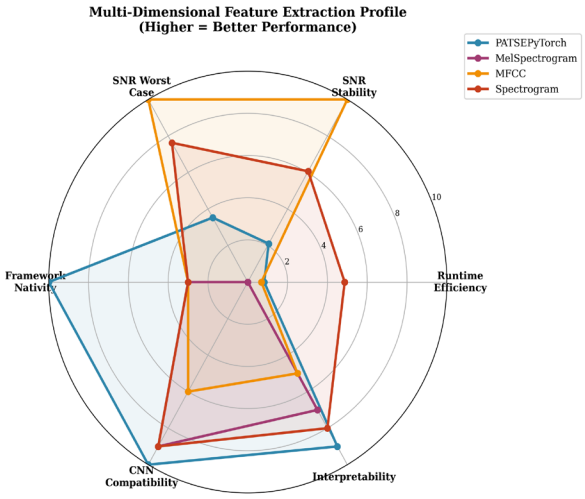


Figure 9. Multi-dimensional feature extraction quality profile.

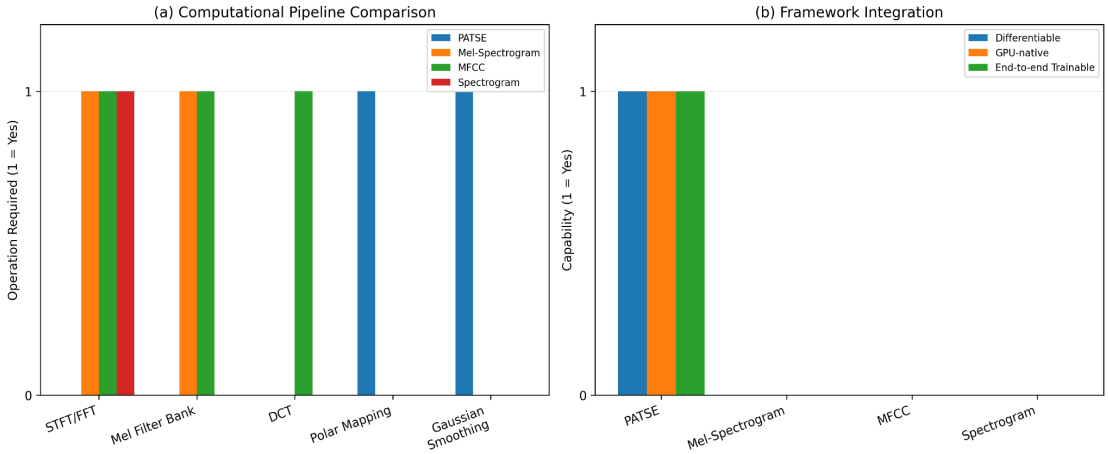


Figure 10. Computational pipeline comparison: (a) operations required by each method (b) framework integration advantages of PATSE over librosa-based baselines

Figure 10 illustrates that PATSE uses a PyTorch-native processing pipeline without NumPy–PyTorch framework conversion, facilitating integration with GPU-compatible PyTorch workflows.

To assess practical effectiveness, all representations were evaluated using the same lightweight CNN comprising three Conv2D layers (32, 64, and 128 filters; 3×3 kernels; ReLU), max-pooling, a Dense(128) layer, and a five-class Softmax output. Inputs were resized to 128×128×3, and training used Adam with categorical cross-entropy, a batch size of 32, and a training-generator seed of 42.

Table 5. Classification performance comparison

Method	Accuracy	Precision	Recall	Macro F1
PATSE	0.91	0.88	0.86	0.85
MelSpectrogram	0.89	0.85	0.83	0.82
MFCC	0.84	0.8	0.78	0.77
Spectrogram	0.87	0.83	0.81	0.8

Table 5 indicates that PATSE retains discriminative information relevant to the downstream classification task while maintaining competitive computational characteristics within the evaluated lightweight CNN setting.

3.2 Discussions

PATSE can be interpreted as a unified computational and representational framework. It employs a fully PyTorch-native pipeline comprising amplitude normalization, coordinate transformation, scatter projection, and separable smoothing, thereby avoiding NumPy–PyTorch framework-conversion steps in the evaluated pipeline. Representationally, PATSE encodes temporal progression and amplitude into spatial geometric patterns. Scatter rendering and Gaussian smoothing produce coherent local spatial structures. Importantly, PATSE is a temporal-amplitude geometric encoding rather than a reconstruction of Fourier spectral phase, and the reported runtime and classification performance should be interpreted as properties of the complete pipeline, since the individual contribution of each processing stage was not evaluated independently.

Extending our previous classification-focused work[12], this study evaluates PATSE from the perspectives of runtime stability, framework integration, and computational efficiency. The original PATSE representation was previously introduced in [12]. Therefore, the novelty of the present study primarily concerns its PyTorch-native implementation and runtime-oriented evaluation**.**

Representational Foundation: Temporal-Amplitude Spatial Encoding. PATSE represents audio as spiral images by jointly encoding temporal progression and waveform amplitude into a spatial geometric structure. This representation provides the foundation for the runtime-oriented implementation and evaluation investigated in the present study [22-28].

Architectural Extension: PyTorch-Native Audio Feature Extraction. PATSE operates within the PyTorch framework, avoiding NumPy conversion and facilitating direct integration with GPU-compatible deep-learning pipelines [29], [30].

Practical Evaluation: Runtime Stability and Computational Efficiency. The present study extends the previous classification-oriented evaluation [12] by examining PATSE in terms of runtime efficiency, execution stability, and framework integration under controlled CPU-based benchmark conditions [31-36].

Spectrogram achieves the fastest runtime, while MFCC exhibits the lowest runtime variability. PATSE occupies an intermediate position in terms of raw runtime and stability, while providing a PyTorch-native representation that facilitates integration with PyTorch-based processing pipelines. Therefore, PATSE should be interpreted as offering a design trade-off among runtime performance, execution stability, and framework integration rather than overall empirical superiority.

Table 6. Multi-dimensional comparative assessment

Dimension	PATSE	MelSpectrogram	MFCC	Spectrogram
Runtime Speed	Moderate (5.37 ms)	Slowest (6.25 ms)	Moderate (5.52 ms)	Fastest (2.56 ms)
Runtime Stability	Moderate (CoV 26.6%)	Poorest (CoV 33.8%)	Best (CoV 12.1%)	Good (CoV 16.6%)
Framework Nativity	(Native PyTorch)	(NumPy bridge)	(NumPy bridge)	(NumPy bridge)
CNN Compatibility	(256×256)	(128×F)	(13×F, narrow)	(513×F)
Visual Representation	(Polar spiral)	Mel-frequency map	Cepstral coefficients	Linear time-frequency map
PyTorch-native	(vectorized)	(Fixed pipeline)	(Fixed pipeline)	(Fixed pipeline)

Table 6 summarizes the multidimensional strengths and limitations of the evaluated methods. Spectrogram provides the fastest runtime, whereas MFCC offers the highest runtime stability. PATSE shows moderate runtime and stability but differs from the conventional baselines through its PyTorch-native implementation and CNN-compatible polar representation. Therefore, PATSE should be interpreted as providing a trade-off among computational efficiency, runtime stability, representation structure, and framework integration rather than overall superiority.

PATSE is designed as a lightweight PyTorch-native front-end representation and is evaluated in terms of computational efficiency, runtime stability, and framework integration, rather than as a replacement for transformer-based audio models such as AST, wav2vec 2.0, HuBERT, and BEATs. Accordingly, these pretrained models were not included in the present experimental benchmark. The empirical comparison

was restricted to Mel-Spectrogram, MFCC, and linear Spectrogram under standardized CPU-based conditions. Consequently, no conclusions regarding PATSE's relative accuracy, computational efficiency, scalability, or deployment suitability compared with pretrained audio models can be drawn from the present results. Future work will investigate direct comparisons with pretrained audio models under appropriately standardized experimental protocols, together with GPU scalability, multi-dataset validation, systematic ablation and sensitivity analyses, and broader classifier robustness.

PATSE exhibits intermediate runtime stability under the evaluated CPU benchmark conditions. Its PyTorch-native design avoids NumPy–PyTorch framework conversion and provides direct integration with PyTorch-based processing pipelines. However, the present results do not establish that PATSE's runtime behavior is caused by a specific representation or architectural component, because individual pipeline stages were not independently evaluated.

Modern pretrained audio models, including AST, wav2vec 2.0, HuBERT, and BEATs, were not experimentally evaluated in this study because the present benchmark focuses on comparing PATSE with conventional front-end audio representations under controlled CPU-based conditions. Consequently, the current findings should not be interpreted as evidence of superiority over pretrained audio representation models. A further limitation is that complete system-level metadata for the original runtime benchmark were not retained, which limits exact replication of the reported absolute latency values across computing environments.

4. Conclusions

This study presented PATSE as an integrated PyTorch-native spatial-temporal encoding framework for infant cry audio. Within the controlled CPU-based evaluation, PATSE achieved competitive runtime performance, more controlled latency behavior than Mel-Spectrogram, and useful discriminative performance under the evaluated lightweight CNN setting. These findings should be interpreted within the scope of the present design. They do not constitute empirical evidence of GPU acceleration, architecture-independent superiority, cross-domain generalizability, or the independent effectiveness of individual PATSE components. GPU scalability, component-level contributions, multi-domain robustness, comparisons with modern pre-trained audio models, and validation across different neural architectures require distinct experimental designs and will be addressed through dedicated follow-up studies.

Declarations

AI Usage Statement

The authors used OpenAI solely to assist with language editing and manuscript revision. All scientific content, data analysis, interpretation, and conclusions are the authors' own.

Author Contributions

N.G.S: conceptualization, methodology, software, formal analysis, visualization, writing and editing – original draft. E.U: supervision, validation and review. K: supervision, validation and review. F.W.W: supervision, validation and review.

Funding Statement

This research received no external funding.

Conflict of Interest Statement

The authors declare no conflict of interest.

Ethical Clearance

This study used the publicly available Donate-a-Cry corpus and involved no direct recruitment, intervention, or interaction with human participants. Therefore, no additional ethical clearance was obtained for this secondary-data study.

Data and Code Availability

The data used in this study are from the publicly available Donate-a-Cry corpus. The implementation code is available from the corresponding author upon reasonable request.

References

[1] N. G. Setyoningrum, E. Utami, Kusrini, and F. W. Wibowo, "A Systematic Literature Review of Audio Signal Processing Methods for Infant Cry Recognition and Interpretation," *CENiM*, 2024, doi: 10.1109/CENIM64038.2024.10882830.

[2] T. Ozseven, "Infant cry classification by using different deep neural network models and hand-crafted features," *Biomed. Signal Process. Control*, vol. 83, p. 104648, 2023, doi: 10.1016/j.bspc.2023.104648.

[3] R. Jahangir, "CNN-SCNet: A CNN net-based deep learning framework for infant cry detection in household setting," *Engineering Reports*, 2023, doi: 10.1002/eng2.12786.

[4] N. G. Setyoningrum, E. Utami, K. Kusrini, and F. W. Wibowo, "Improving Infant Cry Recognition Using MFCC And CNN-Based Audio Augmen tation," *Jurnal Teknik Informatika (Jutif)*, 2025, doi: 10.52436/1.jutif.2025.6.2.4373.

[5] Z. Firas, A. A. Nashaat, and G. Ahmad, "Optimizing Infant Cry Recognition: A Fusion of LPC and MFCC Features in Deep Learning Models," in *International Conference on Advances in Biomedical Engineering, ICABME*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 232–237. doi: 10.1109/ICABME59496.2023.10293083.

[6] H. Choi, L. Zhang, and C. Watkins, "Dual representations: A novel variant of Self-Supervised Audio Spectrogram Transformer with multi-layer feature fusion and pooling combinations for sound classification," *Neurocomputing*, vol. 623, Mar. 2025, doi: 10.1016/j.neucom.2025.129415.

[7] A. S. Podda, R. Balia, L. Pompianu, S. Carta, G. Fenu, and R. Saia, "CARgram: CNN-based accident recognition from road sounds through intensity-projected spectrogram analysis," *Digital Signal Processing: A Review Journal*, vol. 147, Apr. 2024, doi: 10.1016/j.dsp.2024.104431.

[8] N. G. Setyoningrum, E. Utami, Kusrini, and F. W. Wibowo, "A Comprehensive Survey of Infant Cry Classification Research Trends and Methods: A Systematic Review," in *2024 6th International Conference on Cybernetics and Intelligent System*

(ICORIS), IEEE, Nov. 2024, pp. 1–6. doi: 10.1109/ICORIS63540.2024.10903693.
[9] T. Mary Little Flower, T. Jaya, and S. Christopher Ezhil Singh, "Data augmentation using a 1D-CNN model with MFCC/MFMC features for speech emotion recognition," *Automatika*, vol. 65, no. 4, pp. 1325–1338, 2024, doi: 10.1080/00051144.2024.2371249.
[10] K. R. Mannem, E. Mengiste, S. Hasan, B. G. de Soto, and R. Sacks, "Smart audio signal classification for tracking of construction tasks," *Autom. Constr.*, vol. 165, Sep. 2024, doi: 10.1016/j.autcon.2024.105485.
[11] Y. Zayed, A. Hasasneh, and C. Tadj, "Infant Cry Signal Diagnostic System Using Deep Learning and Fused Features," *Diagnostics*, vol. 13, no. 12, p. 2107, 2023, doi: 10.3390/diagnostics13122107.
[12] N. Ghurroh Setyoningrum, E. Utami, F. Wahyu Wibowo, "Polarized Amplitude Time Spiral Encoding for Infant Cry Audio Augmentation and Convolutional Neural Network *Journal of Innovation Information Technology and Application*, vol. 7, no. 2, pp. 379–394, 2025, doi: 10.35970/jinita.v7i2.2960.
[13] M. W. Lakdari, A. H. Ahmad, S. Sethi, G. A. Bohn, and D. J. Clink, "Mel-frequency cepstral coefficients outperform embeddings from pre-trained convolutional neural networks under noisy conditions for discrimination tasks of individual gibbons," *Ecol. Inform.*, vol. 80, p. 102457, 2024, doi: 10.1016/j.ecoinf.2023.102457.
[14] G. Coro, S. Bardelli, A. Cuttano, R. T. Scaramuzzo, and M. Ciantelli, "A self-training automatic infant-cry detector," *Neural Comput. Appl.*, vol. 35, no. 11, pp. 8543–8559, Apr. 2023, doi: 10.1007/s00521-022-08129-w.
[15] A. Abbaskhah, H. Sedighi, and H. Marvi, "Infant cry classification by MFCC feature extraction with MLP and CNN structures," *Biomed. Signal Process. Control*, vol. 86, Sep. 2023, doi: 10.1016/j.bspc.2023.105261.
[16] Y. Gong, Y. A. Chung, and J. Glass, "Ast: Audio spectrogram transformer," in *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, International Speech Communication Association, 2021, pp. 56–60. doi: 10.21437/Interspeech.2021-698.
[17] W.-N. Hsu, B. Bolte, Y.-H. H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, "HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units," Jun. 2021, [Online]. Available: http://arxiv.org/abs/2106.07447. doi.org/10.1109/TASLP.2021.3122291
[18] K. Zaman, K. Li, M. Sah, C. Direkoglu, S. Okada, and M. Unoki, "Transformers and audio detection tasks: An overview," Mar. 01, 2025, *Elsevier Inc.* doi: 10.1016/j.dsp.2024.104956.
[19] X. Li and Z. Zhang, "Cross-feature fusion speech emotion recognition based on attention mask residual network and Wav2vec 2.0," *Digital Communications and Networks*, vol. 11, no. 5, pp. 1567–1577, Oct. 2025, doi: 10.1016/j.dcan.2024.10.007.
[20] J. Cai, Y. Song, J. Wu, and X. Chen, "Voice Disorder Classification Using Wav2vec 2.0 Feature Extraction," *Journal of Voice*, 2024, doi: 10.1016/j.jvoice.2024.09.002.
[21] W. Chen *et al.*, "Classifying Heart-Sound Signals Based on CNN Trained on MelSpectrum an d Log-MelSpectrum Features," *Bioengineering*, vol. 10, no. 6, p. 645, 2023, doi: 10.3390/bioengineering10060645.
[22] T. Guo *et al.*, "SPIRAL: integrating and aligning spatially resolved transcriptomics da ta across different experiments, conditions, and technologies," *Genome Biol.*, 2023, doi: 10.1186/s13059-023-03078-6.
[23] A. Piemontese, A. Ugolini, M. Morini, G. Colavolpe, and T. Eriksson, "Spiral Constellations for Nonlinear Channels," *IEEE Communications Letters*, vol. 28, no. 9, pp. 2016–2020, 2024, doi: 10.1109/lcomm.2024.3438830.
[24] B. Wang, T. Cai, X. Cai, W. Zhou, and Z. Liu, "Polarized imaging dynamic light scattering for simultaneous measurement of nanoparticle size and morphology," *Particuology*, vol. 85, pp. 213–223, Feb. 2024, doi: 10.1016/j.partic.2023.06.004.
[25] G. M. Moatimid and T. S. Amer, "Dynamical system of a time-delayed ϕ6-Van der Pol oscillator: a non-perturbative approach," *Sci. Rep.*, vol. 13, no. 1, 2023, doi: 10.1038/s41598-023-38679-5.
[26] Y. Jing and X. Zhao, "DQ-Former: Querying Transformer with Dynamic Modality Priority for Cognitive-aligned Multimodal Emotion Recognition in Conversation," in *MM 2024 - Proceedings of the 32nd ACM International Conference on Multimedia*, Association for Computing Machinery, Inc, Oct. 2024, pp. 4795–4804. doi: 10.1145/3664647.3681599.
[27] A. Laguna *et al.*, "How can cry acoustics associate newborns' distress levels with neurophysiological and behavioral signals?," *Front. Neurosci.*, 2023, doi: 10.3389/fnins.2023.1266873.
[28] S. Lahmiri, C. Tadj, C. Gargour, and S. Bekiros, "Optimal tuning of support vector machines and k-NN algorithm by using Bayesian optimization for newborn cry signal diagnosis based on audio signal processing features," *Chaos Solitons Fractals*, vol. 167, Feb. 2023, doi: 10.1016/j.chaos.2022.112972.
[29] S. A. Patel and A. Yildirim, "Non-stationary neural signal to image conversion framework for image-b ased deep learning algorithms," *Front. Neuroinform.*, vol. 17, 2023, doi: 10.3389/fninf.2023.1081160.
[30] N. Chelimilla, V. Chinthapenta, and S. Korla, "Addressing data scarcity using audio signal augmentation and deep learning for bolt looseness prediction," *Smart Mater. Struct.*, vol. 33, no. 8, p. 085012, 2024, doi: 10.1088/1361-665x/ad5c24.
[31] K. Ashwini, P. M. D. R. Vincent, K. Srinivasan, and C. Y. Chang, "Deep Learning Assisted Neonatal Cry Classification via Support Vector Machine Models," *Front. Public Health*, vol. 9, Jun. 2021, doi: 10.3389/fpubh.2021.670352.
[32] M. Zhang, Y. Zheng, X. Maidaiti, B. Liang, Y. Wei, and F. Sun, "Integrating Machine Learning into Statistical Methods in Disease Risk Prediction Modeling: A Systematic Review," *Health Data Science*, vol. 4, 2024, doi: 10.34133/hds.0165.
[33] J. Allgaier, L. Mulansky, R. Draelos, and R. Pryss, "How does the model make predictions? A systematic literature review on the explainability power of machine learning in healthcare," *Artif. Intell. Medicine*, 2023, doi: 10.1016/j.artmed.2023.102616.
[34] S. Lee, C. Yin, and P. Zhang, "Stable clinical risk prediction against distribution shift in electron ic health records," *Patterns*, 2023, doi: 10.1016/j.patter.2023.100828.
[35] A. Ayari, H. Hamdi, and K. A. Alsulbi, "E-health Application in IoMT Environment Deployed in An Edge and Cloud Computing Platforms," in *Procedia Computer Science*, Elsevier B.V., 2024, pp. 1019–1028. doi: 10.1016/j.procs.2024.09.521.
[36] T. Hama *et al.*, "Enhancing Patient Outcome Prediction Through Deep Learning with Sequential Diagnosis Codes from Structured Electronic Health Record Data: Sy stematic Review," *J. Med. Internet Res.*, vol. 27, p. e57358, 2025, doi: 10.2196/57358.