Speech Deepfakes Detection with Fast Fourier Transform using Complex Linear Algebra

Kevin Wirya Valerian - 13524019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
kevin.wirya.valerian@gmail.com 13524019@std.stei.itb.ac.id

***Abstract*—Nowadays, the growing realism of speech deepfakes demands detection methods grounded in fundamental signal theory rather than purely data-driven models. A well-known speech deepfake detection approach is based on the Fast Fourier Transform (FFT), rooted in complex linear algebra and geometric signal representations. Speech signals are modeled as vectors in complex vector spaces, where the FFT acts as a structured linear transformation projecting time-domain data onto orthonormal bases of complex exponentials. From a geometric perspective, genuine and synthetic speech can be represented differently in high-dimensional spectral spaces. Simple inconsistencies in phase behavior and energy distribution can be analyzed through algebraic measures derived from the complex FFT representation. This study demonstrates the applicability of FFT-based complex spectral analysis for distinguishing AI-generated and genuine speech.**

***Index Terms*—Fast fourier transform, deepfakes detection, complex linear algebra, spectral geometry**

I. Introduction

Recent advances in speech synthesis and voice conversion have enabled machines to generate highly realistic human speech. Modern text-to-speech and voice cloning systems are capable of producing audio that is increasingly difficult to distinguish from genuine human recordings. A study reports that human participants were only able to accurately distinguish real from AI-generated voices with an accuracy of 70.4%. This leads to a serious risks that can be posed by speech deepfakes in areas such as biometric authentication and digital forensics. Reports indicate that voice-based fraud has increased significantly in recent years, with financial losses reaching billions of dollars globally. Deepfake-related losses have already reached $1.56 billion, with over $1 billion occurring in 2025 alone.

While deep learning models have achieved impressive performance in generating natural-sounding speech, detecting such synthetic audio remains a challenging task. Many existing detection methods rely heavily on large neural networks trained on specific datasets. Although effective, these approaches often lack interpretability and tend to degrade when exposed to unseen deepfake generation methods. Moreover, purely data-driven models can be computationally expensive and difficult to analyze, making them less suitable and reliable at present time.

Speech signals, however, are fundamentally mathematical objects. A digital audio signal can be represented as a finite sequence of samples, which naturally forms a vector in a high-dimensional space. Transforming this signal into the frequency domain using the Fast Fourier Transform (FFT) reveals its spectral structure, including phase behavior. These properties are governed by well-established principles of linear algebra and geometry, such as vector spaces, orthonormal bases, inner or dot products, and unitary transformations. From a linear algebra perspective, the FFT can be interpreted as a linear transformation that maps a time-domain signal into a complex frequency-domain vector. While synthetic speech models often reproduce spectral magnitudes effectively, differences in phase behavior may still arise.

Motivated by this observation, this paper proposes a speech deepfake detection approach grounded in the linear algebraic and geometric interpretation of the FFT. Rather than relying on large data-driven models, the proposed method focuses on analyzing basic algebraic properties of complex spectral representations. The approach aims to provide an interpretable demonstration of how concepts from complex linear algebra and geometry can be applied to distinguish genuine and synthetic speech signals.

II. Theoretical Framework

A. Complex Numbers and Geometric Representation

A complex number is defined as

$$z = a + jb, \quad a, b \in \mathbb{R} \tag{1}$$

where a is the real part and b is the imaginary part, with $j = \sqrt{-1}$. Both i and j represent the same imaginary unit. We use j since it is commonly used for signal processing.

Complex numbers admit a geometric interpretation in the complex plane, where the real part corresponds to the horizontal axis and the imaginary part to the vertical axis. This representation allows us to visualize complex numbers as vectors emanating from the origin.

Every non-zero complex number can be expressed in polar form

$$z = re^{j\theta} \tag{2}$$

where

- $r = |z| = \sqrt{a^2 + b^2}$ is the **magnitude**, representing the distance from the origin

- $\theta = \arg(z) = \arctan(b/a)$ is the **phase**, representing the angle from the positive real (horizontal) axis

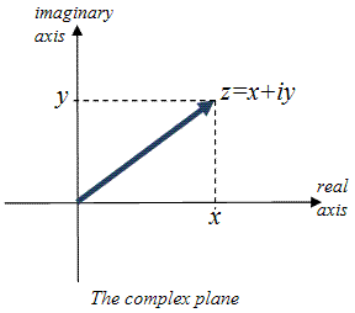

Fig. 1. Argand Plane (Complex Plane)
(Source: https://helpingwithmath.com/complex-plane/)

The exponential form relates to trigonometry via Euler's formula.

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{3}$$

The FFT spectrum of audio signals consists of complex-valued coefficients. Each coefficient X_k can be decomposed into magnitude and phase components. The magnitude $|X_k|$ represents the energy at frequency bin k, while the phase $\angle X_k$ encodes temporal structure. A frequency bin is a specific range on the frequency axis used to group and analyze data. Our detection method exploits the observation that human speech exhibits **regular** phase structure, whereas AI-generated speech tends to have **irregular** phase structure.

B. Complex Vector Spaces $\mathbb{C}^N$

The set $\mathbb{C}^N$ of all N-tuples of complex numbers forms a vector space over the field $\mathbb{C}$. A vector $\mathbf{x} \in \mathbb{C}^N$ is written as

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix}, \quad x_n \in \mathbb{C} \tag{4}$$

Digital audio signals are naturally elements of $\mathbb{R}^N$ in the time domain and $\mathbb{C}^N$ in the frequency domain after FFT transformation. The standard inner product on $\mathbb{C}^N$ is defined as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{n=0}^{N-1} x_n \overline{y_n} \tag{5}$$

where $\overline{y_n}$ denotes the complex conjugate of y_n. This inner product induces the Euclidean norm.

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\sum_{n=0}^{N-1} |x_n|^2} \tag{6}$$

The squared norm $\|\mathbf{x}\|^2$ represents the total energy of the signal.

C. Discrete and Fast Fourier Transform

The Discrete Fourier Transform (DFT) converts a time-domain signal $\mathbf{x} \in \mathbb{C}^N$ to its frequency-domain representation $\mathbf{X} \in \mathbb{C}^N$ via:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N}, \quad k = 0, 1, \ldots, N-1 \tag{7}$$

This transformation can be interpreted as computing the inner product of the signal with complex exponential basis vectors at each frequency k. The inverse DFT reconstructs the time-domain signal:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j2\pi kn/N}, \quad n = 0, 1, \ldots, N-1 \tag{8}$$

The naive DFT requires $O(N^2)$ complex multiplications. The Fast Fourier Transform (FFT) reduces this to $O(N \log N)$ using a divide-and-conquer strategy. The key insight is that DFT can be decomposed into smaller DFTs of even and odd indexed elements.

Let $\omega_N = e^{-j2\pi/N}$ be the primitive N-th root of unity. For $N = 2^m$, we split the DFT:

$$X_k = \sum_{n=0}^{N-1} x_n \omega_N^{kn} \tag{9}$$

$$= \sum_{n=0}^{N/2-1} x_{2n} \omega_N^{k(2n)} + \sum_{n=0}^{N/2-1} x_{2n+1} \omega_N^{k(2n+1)} \tag{10}$$

$$= \sum_{n=0}^{N/2-1} x_{2n} (\omega_N^2)^{kn} + \omega_N^k \sum_{n=0}^{N/2-1} x_{2n+1} (\omega_N^2)^{kn} \tag{11}$$

Since $\omega_N^2 = \omega_{N/2}$, this becomes

$$X_k = E_k + \omega_N^k O_k \tag{12}$$

where E_k is the DFT of even-indexed elements and O_k is the DFT of odd-indexed elements, both of size $N/2$.

Using the symmetry property $X_{k+N/2} = E_k - \omega_N^k O_k$, we compute both halves with a single recursion:

$$X_k = E_k + \omega_N^k O_k, \quad k = 0, \ldots, N/2-1 \tag{13}$$

$$X_{k+N/2} = E_k - \omega_N^k O_k, \quad k = 0, \ldots, N/2-1 \tag{14}$$

The recursion bottoms out at $N = 1$, where $X_0 = x_0$. The total complexity satisfies:

$$T(N) = 2T(N/2) + O(N) = O(N \log N) \tag{15}$$

The FFT preserves several important properties crucial for our detection method:

1. Energy Preservation (Parseval's Theorem)

$$\sum_{n=0}^{N-1} |x_n|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X_k|^2 \tag{16}$$

This ensures that total signal energy is conserved between time and frequency domains.

2. Linearity

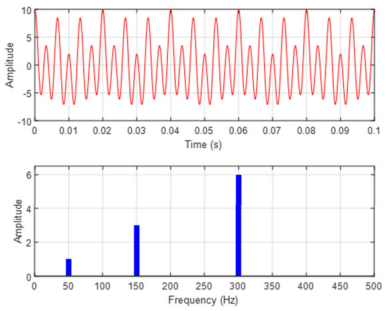

Fig. 2. Time to Frequency Mapping By FFT (Complex Plane) (Source: https://www.sciencedirect.com/topics/engineering/fast-fourier-transform)

$\text{FFT}(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha\text{FFT}(\mathbf{x}) + \beta\text{FFT}(\mathbf{y})$, allowing us to analyze signal components independently.

3. Symmetry for Real Signals:

When $x_n \in \mathbb{R}$, we have $X_{N-k} = \overline{X_k}$, so only the first $N/2$ coefficients need to be stored.

For our implementation, audio signals are real-valued in the time domain, so we extract magnitude and phase from the positive frequency components $k = 0, \ldots, N/2 - 1$.

D. Magnitude and Phase as Complex Algebraic Objects

Each FFT coefficient admits the polar decomposition:

$$X_k = |X_k| e^{j\theta_k} \tag{17}$$

where:

- $|X_k| = \sqrt{\text{Re}(X_k)^2 + \text{Im}(X_k)^2}$ is the spectral magnitude
- $\theta_k = \arctan\left(\frac{\text{Im}(X_k)}{\text{Re}(X_k)}\right)$ is the spectral phase

Phase coherence measures the consistency of phase relationships across frequency bins. For a phase spectrum $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_{N-1})$, we define

$$C(\boldsymbol{\theta}) = \left| \frac{1}{N} \sum_{k=0}^{N-1} e^{j\theta_k} \right| \tag{18}$$

This metric ranges from 0 to 1:

- $C = 1$: Perfectly coherent
- $C = 0$: Completely random
- $0 < C < 1$: Partial coherence

The phase velocity (or phase derivative) measures the smoothness of phase progression

$$v_k = \theta_{k+1} - \theta_k \pmod{2\pi} \tag{19}$$

The variance of phase velocity indicates regularity:

$$\sigma_v^2 = \text{Var}(v_k) \tag{20}$$

Low variance suggests smooth and natural phase evolution, while high variance indicates abrupt phase changes typical of synthesis artifacts.

In our implementation, we quantify phase smoothness using the mean absolute phase gradient:

$$\bar{v} = \frac{1}{N-1} \sum_{k=0}^{N-2} |v_k| \tag{21}$$

where smaller values of $\bar{v}$ indicate smoother phase transitions characteristic of natural speech, while larger values suggest the discontinuous phase patterns often present in synthesized audio.

Modern AI voice synthesis (neural vocoders, GANs) primarily optimizes for perceptually accurate magnitude spectra because human hearing is more sensitive to magnitude than phase. However, these models often fail to produce coherent phase structure because phase reconstruction is mathematically ill-conditioned.

E. Spectral Entropy and Energy Distribution

Spectral entropy quantifies the concentration of energy across the frequency spectrum. For a magnitude spectrum $\mathbf{M} = (|X_0|, |X_1|, \ldots, |X_{N-1}|)$, we first normalize to obtain a probability distribution

$$p_k = \frac{|X_k|}{\sum_{i=0}^{N-1} |X_i|} \tag{22}$$

The spectral entropy is then defined as

$$H(\mathbf{M}) = -\sum_{k=0}^{N-1} p_k \log p_k \tag{23}$$

This metric characterizes the distribution of spectral energy

- **Low entropy**: Energy concentrated in few frequency bins (typical of human speech)
- **High entropy**: Energy spread uniformly across frequencies (may indicate synthesis artifacts)

The spectral L2 norm

$$\|\mathbf{M}\|_2 = \sqrt{\sum_{k=0}^{N-1} |X_k|^2} \tag{24}$$

represents the total energy in the signal and serves as a normalization factor for geometric distance computations.

F. Window-Based Phase Coherence Analysis

While global phase coherence provides an overall measure, local phase patterns can reveal subtle artifacts. We employ a sliding window approach to compute localized coherence. For a window size w, the local phase coherence at position i is

$$C_i^{(w)} = \frac{1}{w} \left| \sum_{k=i}^{i+w-1} e^{j\theta_k} \right| \tag{25}$$

The overall phase coherence is then the mean of all window coherences:

$$C_{\text{window}} = \frac{1}{N-w} \sum_{i=0}^{N-w-1} C_i^{(w)} \tag{26}$$

This windowed approach offers several advantages:

- Captures local phase consistency patterns
- More robust to isolated phase discontinuities
- Reveals frequency-dependent phase artifacts common in AI synthesis

The window size w is typically chosen based on the expected correlation length of natural speech phase patterns.

G. Data Clustering

In our framework, audio samples form clusters in the complex feature space. Let $\mathcal{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_M\}$ be the set of feature vectors from human speech samples, with centroid

$$\boldsymbol{\mu}_H = \frac{1}{M} \sum_{i=1}^{M} \mathbf{h}_i \tag{27}$$

Similarly, AI-generated samples form a cluster $\mathcal{A}$ with centroid $\boldsymbol{\mu}_A$. The optimal decision boundary lies at the midpoint between centroids:

$$\tau = \frac{\boldsymbol{\mu}_H + \boldsymbol{\mu}_A}{2} \tag{28}$$

For a test sample with phase coherence C, the classification rule is:

$$\text{decision}(C) = \begin{cases} \text{HUMAN} & \text{if } C > \tau \\ \text{AI-GENERATED} & \text{if } C \leq \tau \end{cases} \tag{29}$$

where τ is the optimal threshold computed from the training data statistics.

H. Mahalanobis-Like Distance and Confidence Estimation

To quantify classification confidence, we compute standardized distances from the test sample to each cluster centroid. Given the phase coherence C of a test sample, and the statistics (μ_H, σ_H) and (μ_A, σ_A) from the human and AI training sets respectively, we define:

$$d_H(C) = \frac{|C - \mu_H|}{\sigma_H + \epsilon} \tag{30}$$

$$d_A(C) = \frac{|C - \mu_A|}{\sigma_A + \epsilon} \tag{31}$$

where ϵ is a small constant to prevent division by zero. These distances are analogous to Mahalanobis distances in one dimension, accounting for the variance of each class.

The confidence score is computed from the relative distances:

$$\text{confidence} = 1 - \frac{\min(d_H, d_A)}{\max(d_H, d_A) + \epsilon} \tag{32}$$

This confidence metric has the following properties:

- confidence $\in [0, 1]$
- High confidence (≈ 1): Test sample is much closer to one cluster than the other
- Low confidence (≈ 0): Test sample is equidistant from both clusters (ambiguous case)

The predicted class is the one with smaller distance:

$$\text{prediction} = \begin{cases} \text{HUMAN} & \text{if } d_H < d_A \\ \text{AI-GENERATED} & \text{otherwise} \end{cases} \tag{33}$$

This geometric approach provides both a classification decision and a measure of certainty, essential for practical deployment where uncertain predictions may require human review.

III. Methodology

The detection system operates through a four-stage pipeline. Those are audio preprocessing, frequency domain transformation, feature extraction, and threshold-based classification. The core hypothesis is that human speech exhibits distinct phase relationships in the frequency domain that differ systematically from AI-generated audio. These differences arise from fundamental mechanisms that human speech originates from physical vocal cord vibrations and acoustic resonance, while AI-generated speech is synthesized digitally, potentially introducing artifacts in phase structure.

A. Signal Processing

The audio signal is first loaded and normalized to a consistent sampling rate and amplitude range. Stereo recordings are converted to mono by averaging channels. The preprocessed time-domain signal then undergoes Fast Fourier Transform (FFT), converting it from amplitude-over-time representation to amplitude-and-phase-over-frequency representation.

The FFT produces a complex-valued vector where each element corresponds to a frequency component. The magnitude represents energy at that frequency, while the phase angle captures timing relationships. The system retains only positive frequency components, exploiting the symmetry property of real-valued signals. This frequency domain representation becomes the foundation for all subsequent analysis.

B. Feature Extraction

From the FFT output, the system extracts three key features characterizing the signal's geometric properties. First, phase coherence serves as the primary discriminative feature. They are used for quantify how consistently phase transitions across adjacent frequency bins. The computation analyzes phase values in sliding windows across the spectrum. High coherence indicates regular phase transitions typical of human speech. On the other hand, low coherence suggests random or artificial patterns common in nonhuman speech.

Second, phase velocity measures the rate of phase change across frequency. Human speech typically exhibits smoother phase curves due to continuous physical sound production, while AI-generated speech may show abrupt transitions.

Third, spectral entropy quantifies energy distribution across frequencies. Low entropy indicates concentration in specific bands, while high entropy suggests uniform distribution.

C. Classification Strategy

The classification employs a geometric threshold-based approach rather than machine learning. Before classification, the system computes reference statistics from known human and AI-generated speech samples. For each dataset, phase coherence values are extracted from multiple files, and statistical measures (mean and standard deviation) are computed. The decision threshold is established at the midpoint between the human and AI mean phase coherence values.

For a test audio sample, the system extracts its phase coherence and applies a simple decision rule. If phase coherence

exceeds the threshold, it is classified as human. Otherwise, it is classified as nonhuman. To quantify confidence, the system computes normalized geometric distances from the test sample to each class reference, similar to Mahalanobis distance. The distance to each class measures how many standard deviations the test sample deviates from that class's mean. Confidence reflects how much closer the sample is to its predicted class compared to the alternative.

This threshold-based strategy offers immediate interpretability with a clear decision boundary, minimal calibration requirements, and avoidance of overfitting risks. The approach prioritizes mathematical transparency and geometric reasoning over complex learned parameters, making the classification process fully explainable through linear algebra principles.

IV. Implementation

A. Signal Processing

```python
def load_wav(self, filepath):
        try:
            filepath = Path(filepath)
            file_ext = filepath.suffix.lower()
            # Load MP3 files with librosa
            if file_ext == '.mp3':
                if not HAS_LIBROSA:
                    raise ValueError("librosa not installed")
                signal, sr = librosa.load(str(filepath), sr=None, mono=True)
                return signal, sr
            # Load WAV files
            elif file_ext == '.wav':
                sr, signal = wavfile.read(filepath)
                # Convert stereo to mono if needed
                if len(signal.shape) > 1:
                    signal = np.mean(signal, axis=1)
                # Normalize to [-1, 1]
                if signal.dtype in [np.int16, np.int32]:
                    signal = signal.astype(np.float32) / np.max(np.abs(signal))
                else:
                    signal = signal.astype(np.float32)
                return signal, sr
            else:
                raise ValueError(f"Unsupported file format: {file_ext}. Use WAV or MP3.")
        except Exception as e:
            raise ValueError(f"Error loading audio file: {str(e)}")
```

Fig. 3. Loading Audio Files
(Source: Author)

First, audio file input was handled by first detecting the file extension. For WAV files, it uses scipy.io.wavfile.read to load the audio data and metadata, while MP3 files are loaded using the librosa library. The method then applies preprocessing steps. The signal is normalized to a floating-point range of [-1, 1].

```python
def compute_spectral_features(self, signal, sr=None):
        # Apply FFT
        X = fft(signal)
        # Keep only positive frequencies
        N = len(X)
        X = X[:N//2]
        # Compute magnitude and phase in polar form
        magnitude = np.abs(X)
        phase = np.angle(X)
        # Frequency bins (in Hz)
        if sr is None:
            sr = self.target_sr
        freq = np.fft.fftfreq(N, d=1/sr)[:N//2]
        return {
            'X': X,                          # Complex spectral vector
            'magnitude': magnitude,          # Energy spectrum
            'phase': phase,                  # Phase spectrum
            'freq': freq,                    # Frequency axis
            'N': N                           # Original signal length
        }
```

Fig. 4. Computing Mathematical Features of Audio
(Source: Author)

There is a part of the code to transform the time-domain signal into the frequency domain using Fast Fourier Transform (FFT) via scipy.fftpack.fft. All results—complex coefficients, magnitude, phase, and frequency—are organized into a dictionary for downstream feature calculations.

```python
def compute_phase_coherence(self, phase, window_size=5):
        try:
            # Remove NaN and infinite values
            phase = np.nan_to_num(phase, nan=0.0, posinf=0.0, neginf=0.0)
            # Ensure phase is valid
            if len(phase) < window_size:
                return 0.5, np.array([0.5])

            # Compute coherence in sliding windows
            coherence_bins = []
            for i in range(len(phase) - window_size):
                window_phase = phase[i:i+window_size]
                # Sum phase vectors: Σ exp(j*∠X(k))
                phase_sum = np.abs(np.sum(np.exp(1j * window_phase)))
                # Normalize by window size
                coherence = phase_sum / window_size
                coherence_bins.append(coherence)

            # Overall coherence: mean of window coherences
            overall_coherence = np.mean(coherence_bins) if coherence_bins else 0.5
            overall_coherence = np.clip(overall_coherence, 0.0, 1.0)

            return overall_coherence, np.array(coherence_bins)
        except Exception as e:
            return 0.5, np.array([0.5])

    def compute_phase_velocity(self, phase):
        # Phase difference between adjacent frequency bins
        phase_velocity = np.diff(phase)
        phase_velocity = np.angle(np.exp(1j * phase_velocity))
        # Mean absolute velocity / smoothness metric
        velocity_smoothness = np.mean(np.abs(phase_velocity))

        return velocity_smoothness

    def compute_spectral_inner_products(self, magnitude):
        # Euclidean norm of magnitude vector
        l2_norm = np.linalg.norm(magnitude, ord=2)
        # Normalized spectral shape (unit vector for cosine similarity)
        if l2_norm > 0:
            spectral_shape = magnitude / l2_norm
        else:
            spectral_shape = magnitude

        # Spectral entropy
        prob = (magnitude + 1e-10) / (np.sum(magnitude) + 1e-10)
        entropy = -np.sum(prob * np.log(prob))
        return {
            'l2_norm': l2_norm,
            'spectral_shape': spectral_shape,
            'entropy': entropy
        }
```

Fig. 5. Computing Phase Coherence, Phase Velocity, and Spectral Entropy
(Source: Author)

Spectral coherence is computed using a sliding window approach. The method returns both the overall coherence as the mean of all window coherences and an array of per-window values for diagnostic purposes. Next, the the smoothness of phase is quantified by computing the firt-order difference between adjacent phase. Last, there exists a part for extracting L2 norm, then normalize the magnitude vector using it. By using entropy formula, in the end, three metrics are returned in a dictionary. Extraction of all features from a file need to be executed for a complete information gathering.

```python
def extract_all_features(self, filepath):
        # Extracting all features

def extract_features_from_file(filepath):
        # Extracting from file
```

Fig. 6. Extraction
(Source: Author)

B. Reference Statistics

```
def get_wav_files(self, directory):
        audio_files = []
        if os.path.exists(directory):
            audio_files = list(Path(directory).glob('**/*.wav'))
            audio_files += list(Path(directory).glob('**/*.mp3'))
        return sorted(audio_files)
```

Fig. 7. Obtaining WAV Files
(Source: Author)

All audio files inside a certain directory are recursively searched by using glob patterns to find files matching *.wav and *.mp3. The method returns a sorted list of Path objects representing all discovered audio files.

```
def compute_statistics(self, verbose=True):
        stats = {
            'human': {'coherences': [], 'velocities': [], 'entropies': []},
            'nonhuman': {'coherences': [], 'velocities': [], 'entropies': []}
        }
        # Process human speech
        human_files = self.get_wav_files(self.human_dir)
        for filepath in human_files:
            try:
                features = self.processor.extract_all_features(str(filepath))
                stats['human']['coherences'].append(features['phase_coherence'])
                stats['human']['velocities'].append(features['phase_velocity'])
                stats['human']['entropies'].append(features['spectral_entropy'])
            except Exception as e:

        # Process AI-generated speech
        nonhuman_files = self.get_wav_files(self.nonhuman_dir)
        for filepath in nonhuman_files:
            try:
                features = self.processor.extract_all_features(str(filepath))
                stats['nonhuman']['coherences'].append(features['phase_coherence'])
                stats['nonhuman']['velocities'].append(features['phase_velocity'])
                stats['nonhuman']['entropies'].append(features['spectral_entropy'])
            except Exception as e:

        # Compute aggregate statistics
        result = {}

        for label in ['human', 'nonhuman']:
            coherences = np.array(stats[label]['coherences'])
            velocities = np.array(stats[label]['velocities'])
            entropies = np.array(stats[label]['entropies'])
            # Return everything result

        # Print results

        # Compute decision threshold
        mu_h = result['human']['phase_coherence']['mean']
        mu_ai = result['nonhuman']['phase_coherence']['mean']
        threshold = (mu_h + mu_ai) / 2.0

        result['decision_threshold'] = float(threshold)

        # Decision threshold results

        return result
```

Fig. 8. Computing Statistics
(Source: Author)

Baseline statistics must be computed to process human and nonhuman datasets. After processing all files, the method computes aggregate statistics for features such as mean, standard deviation, minimum, and maximum values. Then, the decision threshold is calculated.

```
def compute_and_save_reference_stats(human_dir='../data/human',
                                     nonhuman_dir='../data/nonhuman',
                                     output_file='reference_stats.json'):
    computer = ReferenceStatisticsComputer(human_dir, nonhuman_dir)
    stats = computer.compute_statistics(verbose=True)
    computer.save_statistics(stats, output_file)
    return stats
```

Fig. 9. Computing and Saving Reference Statistics
(Source: Author)

The rest of the code is used to save and load the statistics. When the application is first started, the data will get computed by the functions there.

C. Detector

```
def _compute_geometric_distance(self, phase_coherence):
    mu_h = self.human_stats['mean']
    sigma_h = self.human_stats['std'] + 1e-6  # Avoid division by zero

    mu_ai = self.ai_stats['mean']
    sigma_ai = self.ai_stats['std'] + 1e-6

    # Standardized distances
    d_human = np.abs(phase_coherence - mu_h) / sigma_h
    d_ai = np.abs(phase_coherence - mu_ai) / sigma_ai

    # Confidence computing
    min_dist = min(d_human, d_ai)
    max_dist = max(d_human, d_ai)
    confidence = 1.0 - (min_dist / (max_dist + 1e-6))

    return d_human, d_ai, confidence
```

Fig. 10. Computing Geometric Distance
(Source: Author)

Geometric distance calculation is used for confidence classification. It is derived from the formula mentioned in the theoretical framework. It handles errors and also division by zero.

```
def predict(self, audio_filepath, verbose=False):
    # Extract features
    features = self.processor.extract_all_features(audio_filepath)
    phase_coherence = features['phase_coherence']

    # Decision using threshold
    if phase_coherence > self.threshold:
        primary_prediction = 'human'
    else:
        primary_prediction = 'ai'

    # Compute geometric distances
    d_h, d_ai, confidence = self._compute_geometric_distance(phase_coherence)

    result = {
        'prediction': primary_prediction,
        'confidence': float(confidence),
        'phase_coherence': float(phase_coherence),
        'threshold': float(self.threshold),
        'distance_to_human': float(d_h),
        'distance_to_ai': float(d_ai),
        'phase_velocity': float(features['phase_velocity']),
        'spectral_entropy': float(features['spectral_entropy']),
        'spectral_l2_norm': float(features['spectral_l2_norm'])
    }

    return result

def predict_batch(self, audio_files_list, verbose=False):
    predictions = []
    for filepath in audio_files_list:
        try:
            result = self.predict(filepath, verbose=verbose)
            result['filepath'] = str(filepath)
            predictions.append(result)
        except Exception as e:
            predictions.append({
                'filepath': str(filepath),
                'error': str(e),
                'prediction': None,
                'confidence': None
            })

    return predictions

def get_reference_statistics(self):
    return {
        'human': self.human_stats,
        'ai': self.ai_stats,
        'threshold': self.threshold
    }
```

Fig. 11. Predicting results
(Source: Author)

Prediction performs the main classification task. The method applies pre-computed threshold to classify whether the audio is human or nonhuman. It calculates the confidence score via this data. We used the help of predict_batch to predict multiple audio files.

D. Application

```python
from flask import Flask, request, jsonify, send_from_directory
from flask_cors import CORS
import os
import tempfile
from pathlib import Path
import json

# Detector library
from detector import DeepfakeDetector
from reference_stats import compute_and_save_reference_stats

# Initialize Flask app
app = Flask(__name__)
CORS(app)

# Configuration
UPLOAD_FOLDER = tempfile.gettempdir()
ALLOWED_EXTENSIONS = {'wav', 'mp3'}
REFERENCE_STATS_FILE = 'reference_stats.json'

# Global detector instance
detector = None

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def initialize_detector():
    global detector

    # Check if reference stats exist
    if not os.path.exists(REFERENCE_STATS_FILE):
        print("\nReference statistics not found.")
        print("Computing reference statistics from dataset...")
        print("This may take a few minutes...\n")

        try:
            compute_and_save_reference_stats(
                human_dir='../data/human',
                nonhuman_dir='../data/nonhuman',
                output_file=REFERENCE_STATS_FILE
            )
            print(f"\n[SUCCESS BRO!] Reference statistics saved to {REFERENCE_STATS_FILE}")
        except Exception as e:
            print(f"\n[ERROR BRO!] Failed to compute reference statistics: {str(e)}")
            return False

    try:
        detector = DeepfakeDetector(REFERENCE_STATS_FILE)
        print(f"[SUCCESS] Detector initialized with {REFERENCE_STATS_FILE}")
        return True
    except Exception as e:
        print(f"[ERROR] Failed to initialize detector: {str(e)}")
        return False

# GET and POST methods
# Error handling
# Main Function
```

Fig. 12. Predicting results
(Source: Author)

The Flask web application serves as the user interface for the deepfake detection system. When initializing detector, system automatically checks for the existence of reference statistics and computes them from the training datasets if unavailable, ensuring the system operates well on first launch. Upon receiving a file, the endpoint validates the file format, saves it temporarily to disk, invokes the deepfake detector to perform classification, and returns a JSON response containing the prediction label (human or AI-generated), confidence score, and detailed spectral metrics including phase coherence, geometric distances to both classes, phase velocity, spectral entropy, and L2 norm. Additional endpoints provide system status checks and access to reference statistics for diagnostic purposes.

V. Case Analysis

VI. Conclusion

Attachment

Github: Source Code of ”Speech Deepfakes Detection with Fast Fourier Transform using Complex Linear Algebra”
Here

Youtube Video: Demonstration on The Source Code ”Speech Deepfakes Detection with Fast Fourier Transform using Complex Linear Algebra” using Python
Here

Acknowledgment

The author gratefully acknowledges the blessings and strength granted by God Almighty, which enabled the successful completion of this paper. Sincere appreciation is also extended to Dr. Ir. Rinaldi, M.T., lecturer of the IF2123 Linear and Geometric Algebra course, for his continuous support, guidance, and encouragement throughout the semester and in the writing of this paper. Next, the author would thank both of the author's parents who always give the author supports during ups and downs during the process of this work. Last but not least, the author would also like to thank James Cooley and John Tukey, which significantly inspired this work by popularizing FFT algorithm. It impacts meaningfully to the author competitive programming experience.

References

[1] Ni, Y., et al. (2024). ”A Deepfake Detection Algorithm Based on Fourier Transform of Biological Signal.” Tech Science Press. (Accessed on December 16, 2025) https://www.techscience.com/cmc/v79n3/57116

[2] Mai, K. T., Bray, S., Davies, T., and Griffin, L. D.Warning: Humans cannot reliably detect speech deepfakes.PLOS ONE, 18(8):1–20, 2023. (Accessed on December 19, 2025) https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0285333

[3] Munir, R. (2025). IF2123 Aljabar Linier dan Geometri - Semester I Tahun 2025/2026: Aljabar Kompleks (Update 2024). (Accessed on December 19, 2025) https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-25-Aljabar-Kompleks-2025.pdf

[4] Christensen, M. G. (2019). Introduction to Audio Processing. Springer Nature Switzerland AG. (Accessed on December 19, 2025) https://doi.org/10.1007/978-3-030-11781-8

[5] Burrus, C.S. (1989). Algorithms for Discrete Fourier Transform and Convolution. Springer Science+Business Media New York. (Accessed on December 20, 2025). https://link.springer.com/book/10.1007/978-1-4757-3854-4

[6] Uppada, S. K. (2014). Centroid Based Clustering Algorithms- A Clarion Study. International Journal of Computer Science and Information Technologies, Vol. 5 (6), 7309-7313. (Accessed on December 20, 2025). https://www.ijcsit.com/docs/Volume%205/vol5issue06/ijcsit2014050688.pdf

[7] S. Nordholm, S., Togneri, R., Toh, A. M. (2005). Spectral Entropy as Speech Features for Speech Recognition. PEECS. (Accessed on December 20, 2025). https://www.academia.edu/download/95656320/Spectral_entropy_as_speech_features_for_speech_rec.pdf

[8] Xiang, S., et al. (2008). Learning a Mahalanobis Distance Metric for Data Clustering and Classification. Tsinghua National Laboratory for Information Science and Technology (TNList). (Accessed on December 20, 2025). https://doi.org/10.1016/j.jesp.2017.09.011

Statement

Hereby I declare that this paper that I have written is my own work, not a reproduction or translation of someone else's work and not plagiarized.

Bandung, 24 December 2025

Kevin Wirya Valerian
13524019