soundsafe.ai

Watermarking Model and Commercial Architecture

1. Introduction

This document provides a comprehensive overview of the SoundSafe watermarking model, designed to embed robust and imperceptible watermarks into audio files.

This documentation is intended for the DevOps team to understand the system's architecture, training process, and key components for effective deployment, monitoring, and maintenance.

2. System Architecture

The SoundSafe watermarking model consists of several key components working together:

- **Encoder:**
  - Combines the host audio signal with a message vector containing the watermark data.
  - Embeds watermark information into the audio signal, creating a watermarked audio signal.
- **Shift Module:**
  - Introduces a random shift in the decoding window during training to simulate real-world misalignments.
  - Enhances the robustness of the watermark by training the model to handle these shifts.
- **Attack Simulator:**
  - Simulates common real-world audio distortions (attacks) that a watermarked audio signal might encounter during transmission, storage, or manipulation.
  - Applies one of ten distinct attack types to each training sample during training:
    - Random Noise (RN)
    - Sample Suppression (SS)
    - Low-pass Filter (LP)
    - Median Filter (MF)
    - Re-Sampling (RS)
    - Amplitude Scaling (AS)
    - Lossy Compression (LC) (MP3 Conversion)
    - Quantization (QTZ)
    - Echo Addition (EA)
    - Time Stretch (TS)
- **Decoder:**
  - Extracts the watermark message from the potentially attacked audio signal, aiming to recover the original message vector.
- **Acoustic Fingerprinting:**
  - An approach to generating a fingerprint from both watermarked and unwatermarked audio in order to perform comparisons
- **Database and Querying:**
  - Extracted fingerprints are stored and queried to identify matches

3. Audio Processing Details

- **Audio Input:**
  - Single-channel audio with a sampling rate of 16 kHz.
  - Encoding Unit Length (EUL) is set to 1 second (16,000 samples).
- **Spectrogram Conversion:**
  - Short-Time Fourier Transform (STFT) is applied to convert the audio to the frequency domain.
  - STFT parameters: n_fft=1000, hop_length=400
  - Resultant spectrogram has a shape of (B, 2, W, H), where:
    - B: Batch size
    - 2: Channels (frequency, phase)
    - W: Temporal dimension
    - H: Frequency dimension
- **Inverse STFT (iSTFT):**
  - Used to convert the processed spectrogram back into an audio waveform.

4. Invertible Encoder/Decoder Architecture

- **Encoder:**
  - Consists of 8 invertible blocks.
  - Each invertible block includes three sub-networks:
    - $\rho(\cdot)$: 5-layer 2D CNN with dense connections.
    - $\varphi(\cdot)$: 5-layer 2D CNN with dense connections.
    - $\eta(\cdot)$: 5-layer 2D CNN with dense connections.
  - The encoder concatenates the message vector with the audio spectrogram along the channel dimension before processing with the invertible blocks.
- **Decoder:**
  - Mirror structure of the encoder with inverse operations in reverse order.
  - Extracts the message vector from the processed spectrogram via its inverted processing pipeline.

5. Training Framework

- **Training Process:**
  - Input host audio and message vector to the Encoder.
  - The encoder generates watermarked audio via its inverted neural network.
  - The Shift module randomly shifts the decoding window of the output.
  - The Attack Simulator applies a random attack to the shifted audio.
  - The Decoder attempts to recover the message vector from the distorted audio.
  - The model learns to embed robust watermarks and recover them, based on message recovery success.
- **Loss Functions:**
  - **Message Loss (Lm):** L2 loss between the original and decoded message vector (mvec and m_hat_vec).
  - **Audio Loss (La):** L2 loss between the original and watermarked audio signal (xwave and x'wave).
  - **Discriminator Loss (Ld):** Adversarial loss for the discriminator network to differentiate between original and watermarked audio.
  - **Generator Loss (Lg):** Adversarial loss for the encoder to deceive the discriminator by making the watermarked audio similar to the original audio.
  - **Total Loss (Ltotal):**
    - Ltotal = λaLa + Lm + λgLg
    - λa: Hyperparameter controlling the weight of the audio loss.
    - λg: Hyperparameter controlling the weight of the generator loss.
- **Curriculum Learning Strategy:**
  - **Stage 1:** Focus on learning the core encoding mechanism with weak perceptual constraints and without attacks.
  - **Stage 2:** Introduce attack simulations with relaxed perceptual constraints to enhance robustness.
  - **Stage 3:** Enforce strong perceptual constraints to ensure imperceptibility with attack simulations.
- **Weighted Attack Sampling:**
  - The training process dynamically assigns higher weights to the attacks where the model shows poor performance (higher Bit Error Rate or BER)
  - During training, only one of the ten attacks is applied at a time.
  - Attacks with higher BER have higher sampling weights during the next training iteration.
- **Item-wise Sampling:**
  - Different attack types are applied to different audio samples within the same batch during training, increasing training diversity.
- **Data Augmentation:**
  - Besides attack simulation, data augmentation techniques include random cropping, time stretching, and pitch shifting to improve robustness.

6. Key Technologies

- **Programming Languages:** Primarily Python.
- **Machine Learning Frameworks:** PyTorch.
- **Audio Processing Libraries:** Libraries such as librosa, torchaudio.
- **Database:** SQLAlchemy.
- **Backend:** Flask

7. DevOps Considerations

- **Deployment:**
    - Containerize the model and its dependencies (Docker).
    - Use an orchestration tool (e.g. Kubernetes) to manage containers.
- **Monitoring:**
    - Monitor resource usage (CPU, GPU, memory) of training and inference processes.
    - Track training metrics (loss, BER) for each training epoch.
    - Monitor application logs for any errors or performance issues.
- **Scaling:**
    - The system is designed with scalability in mind through the use of:
        - Batch Processing
        - Parallel Processing
        - Asynchronous Processing
        - Distributed Indexing
        - Microservices
        - API Gateways
    -
    - Distribute training and inference workload across multiple nodes using distributed training capabilities of PyTorch.
- **Continuous Integration/Continuous Deployment (CI/CD):**
    - Establish a robust CI/CD pipeline to automate model training and deployment.
- **API Gateways**
    - The system utilizes API gateways for microservices based architectures.
- **Security:**
    - Implement necessary security measures to protect the audio and watermark data.

Next Steps for the DevOps Team:

1. Review the source code alongside this documentation to familiarize yourself with the codebase.
2. Set up a local development environment and test the model's training and inference workflows.
3. Implement necessary monitoring and logging tools.
4. Contribute to the CI/CD pipeline to automate deployments.
5. Provide any feedback or suggestions on ways to improve the documentation.