Audio-to-MusicXML Converter

The Ultimate Audio Analysis & Transformation Tool

Convert audio files to machine-readable musical notation with extensive transformation capabilities, DAW integration, and comprehensive analysis features.

🎵 Features Overview

- **Multi-format Transcription**: MusicXML, MIDI, ABC notation, LilyPond, CSV
- **DAW Integration**: Generate projects for Ardour, Bitwig, Reaper, Ableton Live
- **Advanced Audio Analysis**: Key detection, chord progressions, tempo analysis
- **Spectral Surgery**: 20+ audio transformation dimensions
- **Batch Processing**: Parallel processing with folder watching
- **Visualizations**: Spectrograms, waveforms, chromagrams
- **Smart Automation**: Presets, random effects, webhook notifications

🚀 Quick Start

Installation

```bash
# Clone and install
git clone [repository-url]
cd audio-to-musicxml-converter

# One-command setup (Ubuntu/Debian)
chmod +x requirements.sh
./requirements.sh

# Manual installation
sudo apt-get install libsndfile1 ffmpeg libaubio-dev cython3
pip install -r requirements.txt
```

Basic Usage

```bash
# Simple transcription
python audio_to_musicxml.py song.wav

# Full analysis with all outputs
python audio_to_musicxml.py song.mp3 --output-all

# Create DAW project
python audio_to_musicxml.py track.flac --send-to-daw ableton
```

📁 Supported Formats

Input: WAV, FLAC, OGG, MP3, MP4, M4A

Output: MusicXML, MIDI, ABC, LilyPond, CSV, JSON

🎛 Audio Transformations

Time & Pitch

- `--time-stretch 1.5` - Stretch duration (1.5x slower)
- `--pitch-shift 7` - Shift pitch (semitones)
- `--formant-shift 3` - Vocal formant shifting

Spectral Processing

- `--spectral-centroid-shift 0.2` - Brightness adjustment
- `--spectral-bandwidth-stretch 1.3` - Frequency spread
- `--frequency-mask-low 100` - High-pass filter (Hz)
- `--frequency-mask-high 8000` - Low-pass filter (Hz)
- `--mel-scale-warp 1.2` - Perceptual frequency warping

Harmonic/Percussive

- `--harmonic-only` - Extract harmonic components
- `--percussive-only` - Extract percussive components
- `--harmonic-percussive-ratio 0.7` - Custom balance (0-1)

Creative Effects

- `--phase-randomize 0.3` - Phase randomization (creates textures)
- `--stutter 0.125` - Rhythmic stuttering (seconds)
- `--granular-synthesis` - Granular processing
- `--dynamic-range-compress 0.6` - Compression ratio

Phase Vocoder

- `--phase-vocoder-stretch 0.8` - High-quality time stretch
- `--tempo-stretch-independent 1.1` - Tempo-aware processing
- `--chroma-shift 2` - Pitch class shifting

🎹 DAW Integration

Generate ready-to-use project files:

```bash
# Ableton Live
python audio_to_musicxml.py song.wav --send-to-daw ableton

# Bitwig Studio
python audio_to_musicxml.py song.wav --send-to-daw bitwig

# Reaper
python audio_to_musicxml.py song.wav --send-to-daw reaper

# Ardour
python audio_to_musicxml.py song.wav --send-to-daw ardour
```

🔍 Advanced Analysis

Musical Analysis

- `--key-detection` - Automatic key detection
- `--chord-analysis` - Chord progression analysis
- `--confidence-threshold 0.8` - Note confidence filtering
- `--pitch-correction` - Snap to nearest semitones

Algorithm Options

- `--pitch-detection-method crepe` - CREPE vs piptrack
- `--tempo-detection-method advanced` - Multi-scale tempo
- `--rhythm-quantize smart` - Intelligent rhythm correction

Quality Control

- `--denoise` - Spectral noise reduction
- `--normalize` - Level normalization
- `--trim-silence` - Remove silence
- `--fade-in 2.0` / `--fade-out 3.0` - Add fades

📊 Visualizations & Reports

```bash
# Generate all visualizations
python audio_to_musicxml.py song.wav --generate-spectrogram --waveform-png --chromagram-image

# Detailed analysis report
python audio_to_musicxml.py song.wav --analysis-report

# Everything at once
python audio_to_musicxml.py song.wav --output-all
```

Generated files:

- `filename_spectrogram.png` - Frequency analysis
- `filename_waveform.png` - Amplitude visualization
- `filename_chromagram.png` - Pitch class analysis
- `filename_report.txt` - Detailed statistics

⚡ Batch Processing

Process Directories

```bash
# Basic batch processing
python audio_to_musicxml.py --batch-dir /music --output-dir /transcriptions

# Parallel processing with effects
python audio_to_musicxml.py --batch-dir /music --output-dir /output --parallel --pitch-shift 2

# Custom file patterns
python audio_to_musicxml.py --batch-dir /music --file-pattern "*.flac" --output-dir /output
```

Folder Monitoring

```bash
# Auto-process new files
python audio_to_musicxml.py --watch-folder /dropbox/music /output --analysis-report
```

🎨 Creative Workflows

Random Experimentation

```bash
# Apply random effects
python audio_to_musicxml.py song.wav --random-transform

# Multiple random variations
for i in {1..5}; do
  python audio_to_musicxml.py song.wav -o variation_$i.musicxml --random-transform
done
```

Preset System

```bash
# Save current settings as preset
python audio_to_musicxml.py song.wav --pitch-shift 7 --time-stretch 0.8 \
  --save-preset "chipmunk" --preset-file presets.json

# Load and apply preset
python audio_to_musicxml.py newsong.wav --preset-name "chipmunk" --preset-file presets.json
```

Effect Chains

bash

```bash
# Complex effect chain
python audio_to_musicxml.py song.wav \
  --pitch-shift 5 \
  --time-stretch 1.2 \
  --harmonic-percussive-ratio 0.8 \
  --phase-randomize 0.2 \
  --spectral-centroid-shift 0.3 \
  --send-to-daw ableton
```

🔧 Configuration & Automation

Config Files

bash

```bash
# Use JSON config file
python audio_to_musicxml.py song.wav --config-file settings.json
```

Webhooks & Integration

bash

```bash
# Webhook notifications
python audio_to_musicxml.py song.wav --webhook-notify http://api.example.com/notify

# With cloud upload
python audio_to_musicxml.py song.wav --upload-to-cloud s3://bucket/path
```

Performance Tuning

bash

```bash
# Custom sample rate
python audio_to_musicxml.py song.wav -sr 48000

# Verbose output with benchmarking
python audio_to_musicxml.py song.wav --verbose --benchmark

# Dry run (test without processing)
python audio_to_musicxml.py song.wav --dry-run
```

📋 Output Formats

MusicXML (Default)

Standard music notation format, readable by most music software and LLMs.

MIDI Export

bash

```bash
python audio_to_musicxml.py song.wav --output-midi
```

Generates `.mid` file for DAW import.

ABC Notation

bash

```bash
python audio_to_musicxml.py song.wav --output-abc
```

Text-based format popular in folk music.

LilyPond

bash

```bash
python audio_to_musicxml.py song.wav --output-lilypond
```

Professional sheet music engraving format.

CSV Data

```bash
python audio_to_musicxml.py song.wav --output-csv
```

Raw note and chord data for analysis.

Everything

```bash
python audio_to_musicxml.py song.wav --output-all
```

Generates all formats plus visualizations and reports.

🎯 Use Cases

Music Transcription

- Convert recordings to sheet music
- Extract MIDI from audio for DAW use
- Analyze musical structure and harmony

Audio Analysis

- Key and tempo detection
- Chord progression analysis
- Spectral content visualization

Creative Processing

- Audio effect experimentation
- Texture creation with phase manipulation
- Harmonic/percussive separation

Batch Workflows

- Process entire music libraries
- Automatic transcription pipelines
- Real-time folder monitoring

LLM Integration

- Generate machine-readable music descriptions
- Feed musical data to AI models
- Automated music analysis

⚙️ Technical Notes

Sample Rate Handling

- Auto-detects input file sample rate by default
- No artificial downsampling unless specified
- Supports upsampling and downsampling
- Saves processed audio when resampling occurs

Performance

- Parallel processing for batch jobs
- Chunked processing for large files
- Optimized algorithms for real-time factors
- Configurable quality vs speed trade-offs

Dependencies

- **Core**: librosa, soundfile, numpy, scipy
- **Visualization**: matplotlib
- **MIDI**: mido
- **Optional**: aubio, essentia, madmom for advanced features

🐛 Troubleshooting

Installation Issues

```bash
# Missing system dependencies
sudo apt-get install libsndfile1 ffmpeg libaubio-dev

# Python package conflicts
pip install --upgrade pip setuptools wheel
```

Audio Quality

- Use higher sample rates (`-sr 44100`) for better pitch accuracy
- Apply `--denoise` for noisy recordings
- Adjust `--confidence-threshold` for cleaner transcriptions

Performance

- Use `--parallel` for batch processing
- Lower sample rates for faster processing
- `--dry-run` to test settings without processing

📄 License

MIT License - Use freely in commercial and personal projects.

🤝 Contributing

This tool uses only open-source libraries and generates patent-free output formats. Contributions welcome for additional features, format support, and performance improvements.

Created with ❤️ for musicians, developers, and AI researchers