# AutoMix & Transcription Engine (ATE) 기술 명세서: 아키텍처 및 구현 로드맵

## 1. 개요 및 프로젝트 범위

### 1.1 서론

현대 디지털 오디오 워크스테이션(DAW)과 AI 기반 음악 처리 기술의 급격한 발전은 음악 제작 및 소비 방식의 패러다임을 변화시키고 있다. **AutoMix & Transcription Engine (ATE)** 프로젝트는 이러한 기술적 진보를 통합하여, 원시 오디오 파일로부터 음악적 구조를 분석(Analysis), 분리(Separation), 재조합(Mixing), 그리고 기호화(Transcription)하는 포괄적인 시스템을 구축하는 것을 목표로 한다. 본 문서는 ATE 시스템의 아키텍처, 데이터 파이프라인, 알고리즘 세부 사항 및 배포 전략을 정의하는 기술 명세서이다.

본 프로젝트의 핵심 개발 철학은 '단계적 완성(Phased Delivery)'이다. 사용자에게 가장 즉각적이고 핵심적인 가치를 제공하는 기능을 우선적으로 개발하여 안정적인 MVP(Minimum Viable Product)를 확보한 후, 고도화된 기능을 추가하는 전략을 채택한다. 이에 따라 개발 로드맵은 다음과 같이 두 단계로 구분된다.

* **Phase 1 (MVP - 핵심 오디오 파이프라인):** 딥러닝 기반의 고품질 음원 분리(Stem Separation), 정밀한 음악 정보 검색(MIR) 기반 오디오 분석, 그리고 신호 처리(DSP) 기술을 활용한 자동 믹싱 시스템의 구현. 이 단계의 완료 시점에서 시스템은 두 곡 이상의 트랙을 분석하고, 조화롭게 믹싱하여 하나의 연속된 오디오 스트림으로 출력할 수 있어야 한다.
* **Phase 2 (기능 확장 - 심볼릭 변환 및 고도화):** 분리된 오디오 소스를 MIDI 데이터로 변환하는 다성 음악 채보(Polyphonic Transcription) 기능과 사용자 경험(UX) 향상을 위한 시각화 및 인터랙티브 기능을 포함한다.

본 문서는 이러한 개발 우선순위에 따라 Phase 1의 기술적 구현을 심도 있게 다루며, 이어지는 Phase 2의 확장 계획을 상세히 기술한다.

### 1.2 시스템 설계 원칙

ATE 시스템은 다음과 같은 핵심 설계 원칙을 준수하여 구축된다.

1. **모듈성(Modularity):** Stem 분리, 분석, 믹싱, 채보 등 각 기능은 독립적인 마이크로서비스 또는 모듈로 설계되어야 한다. 이는 특정 모델(예: Demucs에서 다른 모델로의 교체)이나 알고리즘의 변경이 전체 시스템에 미치는 영향을 최소화하기 위함이다.
2. **확장성(Scalability):** 딥러닝 추론(Inference)은 막대한 연산 자원을 소모한다. 따라서 CPU 집약적인 작업(분석, 렌더링)과 GPU 집약적인 작업(분리, 채보)을 물리적으로 분리하여, 부하에 따라 유연하게 스케일 아웃(Scale-out) 할 수 있는 아키텍처를 지향한다.
3. **데이터 무결성(Data Integrity):** 오디오 데이터는 변환 과정에서 품질 저하가 발생하기 쉽다. 시스템 전반에 걸쳐 고해상도 샘플 레이트(44.1kHz 이상)와 비트 깊이를 유지하며, 믹싱 과정에서의 클리핑(Clipping) 방지 및 라우드니스 정규화(Loudness Normalization)를 필수적으로 수행한다.

## 2. 시스템 아키텍처 및 인프라스트럭처

ATE는 비동기 작업 처리 방식에 기반한 분산 시스템 구조를 채택한다. 사용자의 요청(업로드 및 처리)과 실제 연산 작업(GPU 처리)을 분리함으로써 시스템의 응답성을 보장하고 자원 효율성을 극대화한다.

### 2.1 하이레벨 아키텍처 다이어그램 기술

시스템은 크게 **API Gateway**, **Task Broker**, **Worker Cluster**, **Persistence Layer**의 4가지 계층으로 구성된다.

1. **API Gateway (FastAPI):**
   * 클라이언트와의 유일한 접점 역할을 수행한다. RESTful API를 통해 오디오 파일 업로드, 작업 상태 조회(Polling), 결과물 다운로드 요청을 처리한다.
   * Python의 FastAPI 프레임워크를 사용하여 높은 비동기 처리 성능을 확보한다.1 유효성 검사 및 인증 로직을 수행한 후, 무거운 처리 작업은 Task Broker로 위임한다.
2. **Task Broker (Redis):**
   * 인메모리 데이터 구조 저장소인 Redis를 메시지 브로커로 활용하여 Celery 작업 큐(Job Queue)를 관리한다.
   * 작업의 유형에 따라 큐를 분리한다. 예를 들어, gpu\_queue는 Stem 분리 및 MIDI 변환 작업을, cpu\_queue는 Librosa 분석 및 FFmpeg 렌더링 작업을 처리하도록 라우팅한다.2 이는 고가의 GPU 자원이 CPU 작업으로 인해 유휴 상태(Idle)가 되는 것을 방지하기 위함이다.
3. **Worker Cluster (Celery):**
   * 실질적인 데이터 처리를 담당하는 컨테이너 집합이다.
   * **GPU Worker Node:** NVIDIA CUDA 런타임이 활성화된 환경에서 구동되며, PyTorch 기반의 Demucs 및 Basic Pitch 모델을 로드한다. Docker Compose의 deploy.resources.reservations.devices 설정을 통해 호스트의 GPU에 직접 액세스한다.3
   * **CPU Worker Node:** Numpy, Scipy, Librosa, FFmpeg 등 CPU 연산 위주의 라이브러리를 실행한다.
4. **Persistence Layer (Storage & Database):**
   * **Object Storage (S3/MinIO):** 대용량 오디오 파일(원본, Stem, 믹스 결과물)과 분석 데이터(Spectrogram 이미지, JSON 등)를 저장한다.
   * **PostgreSQL:** 관계형 데이터베이스로, 트랙의 메타데이터, 분석 결과(BPM, Key, Grid 정보), 작업 상태, 사용자 정보를 구조화하여 저장한다.

### 2.2 컨테이너화 및 GPU 가속 전략

딥러닝 모델의 안정적인 배포를 위해 Docker를 사용하며, 특히 GPU 가속을 위한 설정이 필수적이다.

* **NVIDIA Container Toolkit:** 호스트 운영체제에 NVIDIA 드라이버와 Container Toolkit이 설치되어 있어야 하며, Docker 데몬이 nvidia 런타임을 인식하도록 설정해야 한다.4
* **Docker Compose 구성 예시:**  
  Phase 1의 핵심인 Stem 분리를 위해 GPU 워커는 다음과 같이 구성된다.  
  YAML  
  services:  
   worker\_gpu:  
   image: ate-gpu-worker:latest  
   deploy:  
   resources:  
   reservations:  
   devices:  
   - driver: nvidia  
   count: 1  
   capabilities: [gpu]  
   environment:  
   - CELERY\_QUEUE=gpu\_queue  
   - NVIDIA\_VISIBLE\_DEVICES=all  
    
  이 구성은 컨테이너가 호스트의 GPU 자원을 완벽하게 점유하도록 보장하여, Demucs의 복잡한 텐서 연산을 가속화한다.3

## 3. Phase 1: 핵심 오디오 파이프라인 구현

Phase 1은 시스템의 기반이 되는 오디오 처리 파이프라인을 완성하는 단계로, **Stem 분리**, **오디오 분석**, **자동 믹싱**의 세 가지 핵심 모듈로 구성된다.

### 3.1 모듈 1: Stem 분리 엔진 (Demucs)

ATE의 믹싱 엔진은 단순한 주파수 필터링(EQing)이 아닌, 악기별 소스 분리(Source Separation)를 전제로 한다. 이를 통해 베이스라인 교체(Bass Swap)와 같은 고품질 믹싱 기법을 구현한다.

#### 3.1.1 모델 선정 및 아키텍처: Hybrid Demucs

본 프로젝트는 Meta Research의 **Hybrid Demucs (v4)** 아키텍처를 채택한다.5

* **선정 근거:** 기존의 Spleeter나 이전 버전의 Demucs와 달리, Hybrid Demucs는 시간 도메인(Time-domain)의 파형 처리와 주파수 도메인(Frequency-domain)의 스펙트로그램 처리를 결합한 하이브리드 구조를 가진다. 이는 드럼의 타격감(Transient)을 유지하면서도 보컬의 잔향(Reverb)과 같은 주파수 디테일을 보존하는 데 탁월한 성능을 보인다.6
* **분리 대상:** 입력된 오디오는 **Drums**, **Bass**, **Vocals**, **Other**(그 외 반주)의 4개 Stem으로 분리된다.7

#### 3.1.2 기술적 구현 상세

Demucs의 추론 과정은 메모리 사용량이 매우 높으므로, 긴 오디오 트랙을 처리하기 위한 세분화 전략(Segmentation Strategy)이 필요하다.

1. **세그멘테이션(Segmentation):** 전체 오디오를 한 번에 처리할 경우 VRAM 부족(OOM) 현상이 발생할 수 있다. 따라서 오디오를 일정 길이(예: 10초)의 청크(Chunk)로 분할하여 처리한다.5
2. **시프트 및 오버랩(Shifts & Overlap):** 분할된 경계면에서 발생하는 아티팩트(Artifact)를 최소화하기 위해, 각 청크를 겹쳐서 처리(Overlap-add)하고, 시간축을 미세하게 이동시키는 시프트(Shift) 기법을 적용하여 예측의 일관성을 높인다.5
3. **Python API 통합:** demucs.api 또는 torchaudio.pipelines를 래핑하여 사용한다. CLI(Command Line Interface) 방식보다는 Python 스크립트 내부에서 모델 객체를 로드하여 재사용하는 방식이 초기화 오버헤드를 줄이는 데 유리하다.5  
   Python  
   # Demucs 분리 로직 의사 코드 (GPU Worker)  
   import torch  
   from demucs.api import Separator  
     
   def separate\_track(audio\_path, output\_dir):  
    # htdemucs 모델 로드, 세그먼트 길이 12초 설정  
    separator = Separator(model="htdemucs", segment=12, device="cuda")  
     
    # 오디오 파일 분리 실행  
    origin, separated = separator.separate\_audio\_file(audio\_path)  
     
    # 분리된 4개의 Stem을 각각 저장  
    for stem, source in separated.items():  
    # source는 (Channels, Time) 형태의 Tensor  
    save\_wave\_file(source, f"{output\_dir}/{stem}.wav", sr=44100)  
     
   이 코드는 입력된 오디오 파일을 분석하여 4개의 독립된 WAV 파일로 저장하며, 이는 후속 믹싱 과정의 기본 재료가 된다.

### 3.2 모듈 2: 오디오 분석 및 MIR (Librosa)

Stem 분리가 물리적인 소스의 분리라면, 오디오 분석은 음악적 맥락(Context)을 이해하는 과정이다. **Librosa** 라이브러리를 활용하여 BPM, 비트 그리드, 조성(Key), 구조(Structure)를 추출한다.

#### 3.2.1 템포(BPM) 및 비트 추적 (Beat Tracking)

믹싱의 기본은 두 곡의 속도와 박자를 맞추는 것이다.

* **알고리즘:** Librosa의 beat\_track 함수를 사용한다. 이 함수는 온셋(Onset) 강도 엔벨로프를 계산하고, 자기상관(Autocorrelation)을 통해 지배적인 템포를 추정한 뒤, 동적 계획법(Dynamic Programming)을 사용하여 비트의 위치를 확정한다.9
* **정확도 향상:** onset\_envelope를 사전에 계산할 때 aggregate=np.median을 사용하여 일시적인 노이즈에 의한 오류를 줄인다.10

#### 3.2.2 다운비트(Downbeat) 검출 휴리스틱

자동 믹싱을 위해서는 단순한 비트(1/4박) 뿐만 아니라 마디의 시작점인 \*\*다운비트(The "One")\*\*를 아는 것이 필수적이다. Librosa는 기본적으로 다운비트 검출을 지원하지 않으므로, MIR 연구에 기반한 휴리스틱 알고리즘을 구현해야 한다.11

* **구현 로직:**
  1. 대중음악의 표준인 4/4 박자를 가정한다.
  2. beat\_track으로 추출된 비트들을 기반으로, 온셋 강도(Onset Strength) 신호를 비트 단위로 샘플링한다.
  3. 비트 인덱스를 4로 나눈 나머지(Modulo 4)에 따라 4개의 그룹(1박, 2박, 3박, 4박 후보)으로 분류한다.
  4. 각 그룹의 에너지 합계를 계산한다. 통계적으로 마디의 첫 박(다운비트)이 가장 강한 에너지를 가지거나 하모닉 변화가 크다는 특성을 이용한다.
  5. 가장 높은 에너지를 가진 그룹을 다운비트로 판정하고, 이를 기준으로 전체 비트 그리드(Grid)를 재정렬한다.

Python  
# 다운비트 검출 예시 로직  
meter = 4  
# 비트 위치에서의 온셋 강도 추출  
beat\_strengths = onset\_env[beats]   
# 가능한 위상(0, 1, 2, 3)에 대해 에너지 합 계산  
candidates =  
for i in range(meter):  
 energy = np.sum(beat\_strengths[i::meter])  
 candidates.append(energy)  
downbeat\_offset = np.argmax(candidates)  
# 다운비트만 필터링  
downbeats = beats[downbeat\_offset::meter]  
이러한 휴리스틱은 딥러닝 모델(예: Madmom) 없이도 MVP 단계에서 충분히 유효한 마디 정보를 제공한다.11

#### 3.2.3 조성(Key) 감지 및 카멜롯 휠(Camelot Wheel) 매핑

화성적으로 어울리는 믹싱(Harmonic Mixing)을 위해 곡의 조성을 분석한다.

* **크로마그램(Chromagram) 추출:** librosa.feature.chroma\_cqt를 사용하여 오디오의 주파수 성분을 12개의 음계(Pitch Class) 에너지로 변환한다.12 CQT(Constant-Q Transform)는 저주파 대역의 해상도가 높아 음악적 분석에 적합하다.
* **템플릿 매칭:** 추출된 크로마그램의 평균 벡터를 메이저(Major) 및 마이너(Minor) 코드의 이상적인 템플릿과 상관분석(Correlation)하여 가장 유사한 조성을 찾는다.
* **카멜롯 휠 변환:** 분석된 조성(예: A Minor)을 DJ들이 사용하는 **Camelot Wheel** 표기법(예: 8A)으로 변환한다. 이는 인접한 숫자(±1)나 같은 숫자(8A↔8B) 간의 믹싱이 화성적으로 안전하다는 규칙을 시스템화하기 위함이다.14

#### 3.2.4 구조 분할(Structural Segmentation)

곡의 전개(Intro, Verse, Chorus, Outro)를 파악하여 믹싱 포인트를 결정한다.

* **자기유사성 행렬(Recurrence Matrix):** librosa.segment.recurrence\_matrix를 사용하여 곡 내의 반복되는 패턴을 시각화한다.17 대각선 성분이 강하게 나타나는 구간은 동일한 패턴이 반복됨을 의미한다.
* **라플라시안 분할(Laplacian Segmentation):** 자기유사성 행렬에 스펙트럴 클러스터링(Spectral Clustering) 기법을 적용하여 음악적 질감(Texture)이 급격히 변하는 경계점을 검출한다.18 이를 통해 믹싱하기 적합한 구간(예: 드럼이 없는 Intro/Outro)을 자동으로 식별할 수 있다.

### 3.3 모듈 3: 자동 믹싱 엔진 (DSP & FFmpeg)

분석된 데이터와 분리된 Stem을 바탕으로 실제 오디오를 합성하는 엔진이다. Phase 1에서는 **FFmpeg**의 복잡한 필터 그래프(Filter Complex)를 Python으로 제어하여 고품질 믹싱을 구현한다.

#### 3.3.1 믹싱 알고리즘: Bass Swap Transition

EDM 및 팝 음악 믹싱의 표준인 'Bass Swap' 기법을 자동화한다. 두 곡의 저음역대(Kick & Bass)가 겹치면 소리가 뭉개지는 현상(Muddy Mix)을 방지하기 위함이다.

1. **시간 동기화(Time Stretching):** 두 곡의 BPM이 다를 경우, atempo 필터를 사용하여 트랙 B의 속도를 트랙 A에 맞추거나, 목표 BPM으로 두 곡 모두를 변환한다. 피치(Pitch) 변화 없이 속도만 조절해야 한다.19
2. **구간 정렬:** 트랙 A의 믹스 아웃 포인트(예: Outro 시작 16마디 전)와 트랙 B의 믹스 인 포인트(예: Intro 시작)를 다운비트 기준으로 정렬한다.
3. **크로스페이드 로직:**
   * **High/Mid 대역 (Vocals, Other):** 긴 구간(예: 8~16마디)에 걸쳐 부드럽게 크로스페이드(XFade)한다. xfade 필터의 fade 또는 dissolve 트랜지션을 사용한다.19
   * **Low 대역 (Bass, Drums):** 특정 전환점(Swap Point)에서 **하드 컷(Hard Cut)** 하거나 매우 짧은 크로스페이드를 적용한다. 트랙 A의 베이스가 빠지는 동시에 트랙 B의 베이스가 들어오도록 하여 에너지 레벨을 유지한다.

#### 3.3.2 FFmpeg Filter Graph 구성

ffmpeg-python 라이브러리를 사용하여 이러한 복잡한 로직을 그래프 형태로 구성한다.21

Python

import ffmpeg  
  
def create\_mix(track\_a\_stems, track\_b\_stems, swap\_point\_sec, duration):  
 # 입력 스트림 정의  
 a\_bass = ffmpeg.input(track\_a\_stems['bass'])  
 b\_bass = ffmpeg.input(track\_b\_stems['bass'])  
 a\_other = ffmpeg.input(track\_a\_stems['other']) # Drums+Vocals+Other 병합 가정  
 b\_other = ffmpeg.input(track\_b\_stems['other'])  
  
 # 베이스 스왑: 특정 시점(offset)에서 A는 컷아웃, B는 컷인  
 # xfade의 'cut' 트랜지션 사용 (혹은 매우 짧은 duration)  
 bass\_mix = ffmpeg.filter(  
 [a\_bass, b\_bass],  
 'xfade',  
 transition='cut',  
 duration=0.1,  
 offset=swap\_point\_sec  
 )  
  
 # 나머지 악기: 부드러운 전환  
 other\_mix = ffmpeg.filter(  
 [a\_other, b\_other],  
 'xfade',  
 transition='fade',  
 duration=15, # 15초간 오버랩  
 offset=swap\_point\_sec - 7.5 # 스왑 포인트 전후로 페이드  
 )  
  
 # 최종 병합  
 final\_mix = ffmpeg.filter([bass\_mix, other\_mix], 'amix', inputs=2)  
   
 # 렌더링  
 output = ffmpeg.output(final\_mix, 'output\_mix.mp3', audio\_bitrate='320k')  
 ffmpeg.run(output)

이와 같은 필터 그래프는 메모리 상에서 디코딩과 필터링을 파이프라인으로 처리하므로, 중간 파일을 생성하지 않고 효율적으로 믹싱을 수행할 수 있다.22

## 4. Phase 2: 기능 확장 및 MIDI 변환 (상세 계획)

Phase 2는 오디오 신호를 기호(Symbol) 데이터인 MIDI로 변환하고, 사용자 인터페이스를 고도화하는 데 초점을 맞춘다.

### 4.1 모듈 4: 다성 MIDI 변환 (Basic Pitch)

단순한 멜로디(Monophonic)가 아닌, 화음과 여러 악기가 섞인 오디오를 MIDI로 변환하기 위해 Spotify의 **Basic Pitch** 모델을 도입한다.24

#### 4.1.1 Basic Pitch 기술적 특징

* **CQT 기반 신경망:** Basic Pitch는 입력 오디오의 CQT 스펙트로그램을 분석하는 경량화된 신경망을 사용한다.
* **다성(Polyphonic) 지원:** 동시에 여러 음이 연주되는 상황을 인식할 수 있으며, 특히 피치 벤드(Pitch Bend)와 같은 미세한 표현까지 캡처할 수 있는 것이 특징이다.25
* **악기 무관(Instrument Agnostic):** 특정 악기에 과적합되지 않아 다양한 소스에 범용적으로 사용할 수 있다.

#### 4.1.2 Phase 1 결과물 활용 전략

Phase 1에서 분리된 Stem을 활용하면 MIDI 변환의 정확도를 비약적으로 높일 수 있다. 전체 믹스를 변환하는 대신, 분리된 **Bass Stem**과 **Vocal Stem**, **Other Stem**을 각각 Basic Pitch 모델에 입력한다.

* **Bass MIDI:** 베이스라인의 명확한 리듬과 음정을 추출하여 리믹스 제작 시 가이드로 활용.
* **Vocal MIDI:** 보컬 멜로디를 추출하여 신디사이저로 더블링하거나 튜닝하는 데 활용.

#### 4.1.3 구현 및 후처리 (Quantization)

Basic Pitch의 출력은 raw MIDI 이벤트이므로, 음악적 활용을 위해 Phase 1에서 분석한 비트 그리드(Beat Grid)에 맞춰 **퀀타이즈(Quantize)** 하는 후처리 과정이 필요하다.

Python

from basic\_pitch.inference import predict\_and\_save  
  
# 분리된 보컬 Stem을 MIDI로 변환  
predict\_and\_save(  
 audio\_path\_list=['vocals.wav'],  
 output\_directory='./midi\_out',  
 save\_midi=True,  
 onset\_threshold=0.6,  
 frame\_threshold=0.4  
)  
# 이후 생성된 MIDI 파일을 분석된 BPM Grid에 맞춰 스냅(Snap) 처리

### 4.2 사용자 기능 고도화

* **인터랙티브 파형 시각화:** 분석된 세그먼트와 비트 그리드를 웹 UI 상의 파형 위에 오버레이하여 보여준다. 사용자가 드래그 앤 드롭으로 믹싱 포인트(Swap Point)를 수동으로 수정할 수 있는 기능을 제공한다.
* **실시간 미리듣기:** FFmpeg 렌더링 전, 저음질/저지연 모드로 트랜지션 구간만 빠르게 프리뷰하는 기능을 추가한다.

## 5. 데이터 영속성 및 데이터베이스 스키마

시스템의 상태 관리와 분석 데이터의 저장을 위해 관계형 데이터베이스(PostgreSQL)를 사용한다.

### 5.1 데이터베이스 스키마 설계

| **Table Name** | **Description** | **Key Columns** |
| --- | --- | --- |
| tracks | 업로드된 원본 트랙 정보 | id (PK), user\_id, s3\_key, status (processing/ready) |
| analysis\_data | MIR 분석 결과 메타데이터 | track\_id (FK), bpm (float), key (varchar), camelot (varchar), downbeats (jsonb), segments (jsonb) |
| stems | 분리된 오디오 파일 경로 | track\_id (FK), type (enum: drums, bass...), s3\_url |
| mixes | 생성된 믹스 프로젝트 정보 | id (PK), track\_a\_id, track\_b\_id, transition\_params (jsonb), output\_url |

특히 analysis\_data 테이블의 downbeats와 segments 컬럼은 JSONB 타입을 사용하여 가변적인 길이의 배열 데이터(비트 타임스탬프 등)를 효율적으로 저장하고 쿼리할 수 있도록 설계한다.

## 6. 결론 및 기대 효과

본 기술 명세서는 **AutoMix & Transcription Engine (ATE)** 의 성공적인 구축을 위한 청사진을 제시한다.

1. **우선순위의 명확화:** Phase 1에서 Demucs를 통한 고품질 **Stem 분리**, Librosa를 활용한 정교한 **오디오 분석**, 그리고 DSP 기반의 **자동 믹싱**을 완벽하게 구현함으로써, 시스템은 초기 단계부터 사용자에게 실질적이고 독창적인 가치(AI 기반 매시업 및 리믹스 도구)를 제공할 수 있다.
2. **기술적 차별성:** 단순한 크로스페이드를 넘어선 **Bass Swap** 믹싱 기법의 자동화와, 딥러닝 기반의 **다성 MIDI 변환** (Phase 2)은 기존의 상용 서비스들과 차별화되는 ATE만의 핵심 경쟁력이다.
3. **확장 가능한 구조:** GPU/CPU 워커의 분리와 마이크로서비스 아키텍처는 향후 사용자 증가와 트래픽 폭주에 유연하게 대응할 수 있는 견고한 기반을 제공한다.

이 명세서에 따라 개발을 진행함으로써, ATE는 단순한 오디오 처리 도구를 넘어 뮤지션과 크리에이터들의 창작 활동을 보조하는 강력한 AI 파트너로 자리매김할 것이다.

**참고 문헌 및 기술 자료:**

본 보고서는 다음의 기술 문서 및 연구 자료를 바탕으로 작성되었다.

* **Demucs:** Hybrid Transformer Architecture 및 Split/Shift 전략 5
* **Librosa & MIR:** Beat Tracking, Onset Strength, Chromagram, Recurrence Matrix 9
* **DSP & Mixing:** Camelot Wheel 이론, FFmpeg Filter Complex, Crossfade Logic 14
* **Basic Pitch:** CQT 기반 Polyphonic Transcription 24
* **Infrastructure:** Docker GPU Passthrough, Celery/Redis Queueing 1

#### 참고 자료

1. Dockerizing Celery and FastAPI - TestDriven.io, 1월 29, 2026에 액세스, <https://testdriven.io/courses/fastapi-celery/docker/>
2. Docker for AI: GPU Acceleration and Task Queuing with Celery, 1월 29, 2026에 액세스, <https://medium.com/@tokosbex/docker-for-ai-gpu-acceleration-and-task-queuing-with-celery-ae254059b92e>
3. Docker compose equivalent of `docker run --gpu=all` option [closed], 1월 29, 2026에 액세스, <https://stackoverflow.com/questions/70761192/docker-compose-equivalent-of-docker-run-gpu-all-option>
4. Setting up Docker and Docker-Compose with NVIDIA GPU Support ..., 1월 29, 2026에 액세스, <https://medium.com/@jared.ratner2/setting-up-docker-and-docker-compose-with-nvidia-gpu-support-on-linux-716db95c0f7c>
5. demucs/docs/api.md at main - GitHub, 1월 29, 2026에 액세스, <https://github.com/facebookresearch/demucs/blob/main/docs/api.md>
6. Demucs vs Spleeter - The Ultimate Guide - Beats To Rap On, 1월 29, 2026에 액세스, <https://beatstorapon.com/blog/demucs-vs-spleeter-the-ultimate-guide/>
7. Music source separation - Brian Lo - Medium, 1월 29, 2026에 액세스, <https://medium.com/@brinlo/music-source-separation-af7c8ad77d00>
8. demucs - Music Source Separation in the Waveform Domain - PyPI, 1월 29, 2026에 액세스, <https://pypi.org/project/demucs/2.0.0/>
9. Source code for librosa.beat, 1월 29, 2026에 액세스, <https://librosa.org/doc/0.10.2/_modules/librosa/beat.html>
10. librosa.beat.beat\_track — librosa 0.11.0 documentation, 1월 29, 2026에 액세스, <https://librosa.org/doc/main/generated/librosa.beat.beat_track.html>
11. Detecting beat energy with Librosa, finding the first beat of each bar, 1월 29, 2026에 액세스, <https://stackoverflow.com/questions/57384448/detecting-beat-energy-with-librosa-finding-the-first-beat-of-each-bar>
12. Tutorial — librosa 0.10.2 documentation, 1월 29, 2026에 액세스, <https://librosa.org/doc/0.10.2/tutorial.html>
13. Librosa: A Python Audio Libary. by - Medium, 1월 29, 2026에 액세스, <https://medium.com/@patrickbfuller/librosa-a-python-audio-libary-60014eeaccfb>
14. Learn how to use the Camelot Wheel to DJ - Pyramind Institute, 1월 29, 2026에 액세스, <https://pyramind.com/dj-mixing-using-camelot-system-mixed-in-key/>
15. The DJ's Guide to the Camelot Wheel and Harmonic Mixing, 1월 29, 2026에 액세스, <https://dj.studio/blog/camelot-wheel>
16. Master Harmonic Mixing Using the Camelot Wheel for DJs, 1월 29, 2026에 액세스, <https://blog.masterchannel.ai/music-production-tips-using-the-camelot-wheel/>
17. Source code for librosa.segment, 1월 29, 2026에 액세스, <https://librosa.org/doc/0.10.2/_modules/librosa/segment.html>
18. Laplacian segmentation — librosa 0.11.0 documentation, 1월 29, 2026에 액세스, <http://librosa.org/doc/0.11.0/auto_examples/plot_segmentation.html>
19. CrossFade, Dissolve, and other Effects using FFmpeg's xfade Filter, 1월 29, 2026에 액세스, <https://ottverse.com/crossfade-between-videos-ffmpeg-xfade-filter/>
20. A video concatenation tool based on FFMPEG with crossfade ..., 1월 29, 2026에 액세스, <https://gist.github.com/royshil/369e175960718b5a03e40f279b131788>
21. ffmpeg-python documentation - GitHub Pages, 1월 29, 2026에 액세스, <https://kkroening.github.io/ffmpeg-python/>
22. A Guide to Using FFmpeg with Python | by UATeam | Medium, 1월 29, 2026에 액세스, <https://medium.com/@aleksej.gudkov/ffmpeg-python-example-a-guide-to-using-ffmpeg-with-python-020cdb7733e7>
23. FFMPEG & Filter Complex: A Visual Guide to the Filtergraph Usage, 1월 29, 2026에 액세스, <https://www.youtube.com/watch?v=Yc16I6i9xDU>
24. spotify/basic-pitch at 3cf4f08 - Update README.md - Hugging Face, 1월 29, 2026에 액세스, <https://huggingface.co/spotify/basic-pitch/commit/3cf4f083a240dc327d9cd30a3d542424b029b11b>
25. Unlocking Music Creation: Spotify's Basic Pitch Explained - nixsense, 1월 29, 2026에 액세스, <https://xenix.blog/2025/05/06/what-is-spotifys-basic-pitch-and-how-do-you-use-it/>
26. Music Source Separation with Hybrid Demucs - PyTorch, 1월 29, 2026에 액세스, <https://docs.pytorch.org/audio/stable/tutorials/hybrid_demucs_tutorial.html>
27. librosa: Audio and Music Signal Analysis in Python - Brian McFee, 1월 29, 2026에 액세스, <https://brianmcfee.net/papers/scipy2015_librosa.pdf>
28. basic-pitch/basic\_pitch/inference.py at main - GitHub, 1월 29, 2026에 액세스, <https://github.com/spotify/basic-pitch/blob/main/basic_pitch/inference.py>