

ACE-Step Data-Tool

Überblick

Das **ACE-Step Data-Tool** ist ein modulares Tool zur automatisierten Erstellung von Datensätzen für das ACE-Step Musikmodell. Es generiert vollautomatisch annotierte Datensätze aus Audiodaten, die für das Training von Musikmodellen verwendet werden können.

Funktionen

Songtext-Beschaffung (Lyrics Scraping)

Das Tool beschafft automatisch Songtexte zu den vorhandenen Audiodateien aus dem Internet, z. B. von [Genius.com](https://www.genius.com). Falls keine direkte URL verfügbar ist, wird eine Suchanfrage auf Genius durchgeführt und das erste passende Ergebnis verwendet. Die Songtexte werden als reiner Text extrahiert (mittels `Requests` und `BeautifulSoup4`) und in einer Datei namens `__lyrics.txt` gespeichert.

- **Format:** In jeder `__lyrics.txt`-Datei werden Künstler und Titel vermerkt, gefolgt vom vollständigen Liedtext (UTF-8 kodiert).
- **Hinweis:** Zwischen Anfragen wird eine kurze Pause eingelegt, um die Webseite nicht zu überlasten. Falls kein Text gefunden wird, wird der Song als „ohne Lyrics“ markiert.

Tipp: Stellen Sie sicher, dass die Audiodateien korrekte Künstler- und Titel-Tags haben, damit die Lyrics-Erkennung zuverlässig funktioniert.

Tempo-Analyse (BPM-Erkennung)

Das Tool ermittelt das ungefähre Tempo (in BPM – Beats per Minute) aus dem Audiosignal. Dafür wird die Bibliothek `Librosa` verwendet, die Onset-Erkennung und Rhythmusfunktionen nutzt, um das dominante Tempo in den ersten Sekunden des Songs abzuschätzen.

- Extreme Werte werden gefiltert, und Songs ohne erkennbares Tempo werden übersprungen.

Tag-Generierung (Stichwortliste)

Für jeden Track wird eine Liste von Beschreibungstags generiert, die Genres, Stimmung, Instrumentierung, Vocal-Typ etc. beschreiben. Dies geschieht über ein Large Language Model (LLM) mittels einer Ollama-API-Integration.

- **Input:** Künstler, Titel, BPM, ein Auszug des Songtextes (z. B. 300 Zeichen) sowie optionale Stichworte aus Presets oder manuellen Eingaben.
- **Output:** 8-10 Tags in Kleinbuchstaben mit Bindestrich-Notation (z. B. `dark, bass-heavy, female-vocal`), einschließlich eines BPM-Tags (Format: `bpm-XXX`).
- **Regeln:** Maximal 2 Genre-Tags, mindestens je ein Tag für Stimmung und Stil.

Die generierten Tags werden in einer Datei namens `_prompt.txt` gespeichert.

Beispiel:

```
bpm-125, male-vocal, rock, guitars, aggressive, anthem, pop-rock, upbeat, stadium, america
```



ACE-Step Data-Tool

Modulares Tool zur automatisierten Erstellung von Datensätzen für das ACE-Step Musikmodell.
Erstellt aus Audiodateien vollautomatisch begleitende Textdaten (Songtexte/Lyrics und Beschreibungstags)
zur Trainingsdaten-Generierung.

Inhalt

- Übersicht
- Technische Architektur & Hauptfunktionen
- Installation & Setup
- Abhängigkeiten
- Beispiele zur Nutzung
- Modellkompatibilität & Hardware-Anforderungen
- Projektstruktur

Übersicht

ACE-Step Data-Tool ist ein Datenaufbereitungstool für Audiodaten, das speziell entwickelt wurde, um Trainingsdaten für das ACE-Step Musik-KI-Modell automatisch zu erstellen. Es durchsucht einen Ordner voller Audiodateien (z. B. Songs) und generiert für jeden Track zusätzliche Textdateien – etwa die Songtexte (Lyrics) und eine Liste von Stichwörtern/Tags, welche die Stimmung, Genre, Instrumentierung u.a. des Stücks beschreiben. Diese Dateien dienen als Begleitdaten, z. B. um ein KI-Modell wie ACE-Step mit passenden Eingabe-Prompts (Tags + Lyrics) zu trainieren oder um Musikmetadaten zu analysieren.

Hauptziele des Tools sind:

- **Automatisierung** (minimiert manuelle Schritte beim Datenaufbereiten)
- **Modularität** (klare getrennte Komponenten für Lyrics, BPM, Tagging etc.)
- **Benutzerfreundlichkeit** (einfache Nutzung über eine Web-Oberfläche oder Skripte, anpassbar an verschiedene Musikgenres und Modelle)

Das Tool richtet sich an Entwickler:innen und Forschende, die mit Python/Conda und Audiodatensätzen arbeiten, und ermöglicht ein schnelles Erzeugen von beschreibenden Metadaten zu einer Musiksammlung – ohne veraltete, manuelle Workflows.

Technische Architektur & Hauptfunktionen

Das ACE-Step Data-Tool besteht aus mehreren Komponenten, die in einer Verarbeitungspipeline zusammenwirken, unterstützt durch eine optionale Gradio Web-Oberfläche für komfortable Bedienung:

1. **Verzeichnisscan & Dateiverarbeitung:** Standardmäßig erwartet das Tool einen Ordner (`data/` im Projektverzeichnis) mit Audiodateien (unterstützt: `.mp3`, `.wav`, `.flac`, `.m4a`). Alle Dateien (rekursiv in Unterordnern) werden eingelesen und nacheinander verarbeitet. Für jeden Track werden Zwischenergebnisse und Logs angezeigt.
2. **Metadaten-Extraktion:** Zu Beginn wird aus jeder Audiodatei der Artist und Title ermittelt (via ID3-Tags, mit der Bibliothek `TinyTag`). Diese Informationen dienen später zur Lyrics-Suche. Falls eine Datei keine Tags besitzt (z. B. ungetaggte WAVs), wird der Dateiname als Titel genutzt.
 - **Hinweis:** Stellen Sie sicher, dass die Audiodateien korrekte Künstler- und Titeltags haben, damit die Lyrics-Erkennung zuverlässig funktioniert.
3. **Songtext-Beschaffung (Lyrics Scraping):** Das Tool versucht automatisch die Songtexte zum jeweiligen Stück aus dem Internet abzurufen. Es nutzt dazu `Genius.com` als Quelle, wobei zunächst aus den Künstler-/Titelnamen eine passende Lyrics-URL gebildet wird. Falls die direkte URL nicht

erfolgreich ist, wird eine Suchanfrage auf Genius ausgeführt und das erste passende Ergebnis verwendet. Die Songtexte werden als reiner Text extrahiert (mittels `Requests` + `BeautifulSoup4`) und in einer Datei `<Name>_lyrics.txt` gespeichert.

- **Dateiformat:** In jeder `<Name>_lyrics.txt` werden standardmäßig zunächst Künstler und Titel vermerkt, gefolgt vom vollständigen Liedtext (UTF-8 kodiert).
- **Hinweis:** Zwischen den Anfragen wird eine kurze Pause eingelegt, um die Zielwebsite nicht zu überlasten. Sollte kein Text gefunden werden, bricht das Tool die Verarbeitung des jeweiligen Songs ab (d.h. ohne Lyrics erfolgt keine Tag-Generierung – rein instrumentale Tracks oder unbekannte Songs werden übersprungen mit entsprechendem Hinweis).

4. **Tempo-Analyse (BPM-Erkennung):** Parallel wird aus dem Audiosignal das ungefähre Tempo in BPM (Beats per Minute) ermittelt. Hierfür kommt `Librosa` (Audio-Processing-Bibliothek) zum Einsatz: es werden Onset-Erkennung und Rhythmus-Funktionen genutzt, um das dominante Tempo in den ersten Sekunden des Songs abzuschätzen. Extreme Werte werden in einen sinnvollen Bereich normiert (z. B. BPM über 140 werden halbiert; unter 60 ggf. verdoppelt), um halbe/doppelte Zählweisen auszugleichen. Das Ergebnis (Ganzzahl) wird als BPM-Wert für die Tag-Generierung genutzt. Falls die automatische Erkennung fehlschlägt, versucht das Tool, eine BPM-Zahl aus dem Dateinamen zu lesen (z. B. `songname_bpm120.mp3`). Andernfalls bleibt BPM unbekannt.



5. **Tag-Generierung (Stichwortliste):** Als zentrales Feature erzeugt das Tool pro Track eine Liste von Beschreibungstags, die Genre, Stimmung, Instrumentierung, Vocal-Typ etc. des Songs abbilden. Diese Tags werden mittels einer LLM-basierten KI generiert: Das Tool nutzt ein lokales Sprachmodell (Large Language Model) über eine Ollama-API-Integration, um aus den verfügbaren Informationen passende Stichwörter zu „erfinden“.

- Dabei wird dem Modell ein strukturierter Prompt gegeben, der alle vorhandenen Metadaten liefert: Artist, Title, BPM des Songs, ein Auszug des Songtexts (bis zu ~300 Zeichen) zur inhaltlichen Orientierung, sowie optionale Stilhinweise. Diese Stilhinweise setzen sich entweder automatisch aus vom Benutzer gewählten Einstellungen (siehe Presets/Mood) oder einem manuell eingegebenen Prompt-Zusatz zusammen.
- Zusätzlich erhält das Modell feste Regeln im Prompt, wie die Ausgabe aussehen soll: Es muss 12–14 Tags in Kleinschreibung und mit Bindestrich-Notation erzeugen (z. B. `dark, bass-heavy, female-vocal` etc.), unbedingt einen BPM-Tag einschließen (Format `bpm-xxx`), maximal 2 Genre-Tags und mindestens je einen Tag aus den Kategorien Vocals, Instrumente, Stimmung, Rap-Styles. Eine Beispielsammlung wird im Prompt mitgegeben, damit das Modell den Stil erkennt.
- Als Ergebnis der Modell-Abfrage erhält das Tool eine Zeichenkette mit den Tags. Diese werden nachbereitet und gefiltert (unsinnige oder zu lange Begriffe entfernt, Format vereinheitlicht) und als Liste gespeichert. Außerdem wird sichergestellt, dass der BPM-Tag (falls nicht vorhanden oder an späterer Stelle) eingefügt bzw. an den Anfang sortiert wird.
- Die finale Tag-Liste wird dann in einer Textdatei `<Name>_prompt.txt` gespeichert (Tags kommasepariert in einer Zeile). Diese Datei repräsentiert den Prompt für das ACE-Step Modell und kann z. B. bei der Musikgenerierung oder in einem Datensatz als Beschreibung verwendet werden.

6. **Gradio Web-Oberfläche:** Das Tool bietet eine intuitiv bedienbare Web-UI (basierend auf Gradio), über die alle obigen Schritte mit wenigen Klicks ausgeführt werden können. Beim Start des Tools (siehe Setup unten) öffnet sich ein lokaler Webservice auf `http://127.0.0.1:7860` mit folgendem Funktionsumfang:

- **Datei-Scan & Start:** Ein Klick auf „Start Tagging“ durchsucht den definierten `data/` Ordner und verarbeitet alle gefundenen Audiodateien automatisch wie oben beschrieben. Der Fortschritt wird live im Interface protokolliert (für jeden Song wird angezeigt, ob Lyrics gefunden wurden und ob Tags gespeichert wurden).
- **Lyrics-Option:** Über eine Checkbox „Overwrite lyrics“ kann gesteuert werden, ob vorhandene Lyrics-Dateien überschrieben werden sollen. Ist diese Option deaktiviert, werden bereits existierende `<Name>_lyrics.txt` beibehalten (kein erneutes Herunterladen der Texte), was Zeit spart, wenn man einen Lauf wiederholt.
- **Genre-Presets:** In einem Dropdown „Genre-Preset“ kann ein vordefiniertes Genre/Stil-Preset gewählt werden. Diese Presets liegen als einfache Text- oder JSON-Dateien im Ordner `presets/` und enthalten bestimmte Schlagwörter (z. B. ein Preset „HipHop“ könnte Begriffe wie „90s,

boom bap, underground" vorgeben). Der ausgewählte Preset-Text fließt als Stilvorgabe in den Prompt an das Modell ein, sodass die generierten Tags stilistisch angepasst werden.

- **Hinweis:** Sie können eigene Preset-Dateien in `presets/` ablegen – beim Start werden alle `.txt/.json` dort automatisch geladen.
 - **Mood-Slider:** Über den Schieberegler „ Mood: Sad ↔ Happy“ lässt sich die Stimmungsvorgabe variieren. Ein neutraler Wert (0.0) bedeutet keine explizite Stimmungsvorgabe. Bei positiven Werten (>0.3) wird die Stilvorgabe automatisch um einen “happy”-Tag ergänzt, bei negativen (<-0.3) um “sad”. Dies beeinflusst z. B. die resultierenden Stimmungstags (etwa ob „chill“/„happy“ oder „dark“/„sad“ generiert wird).
 - **Prompt-Zusatz:** Ein Textfeld „ Prompt addition (optional)“ ermöglicht es, manuell zusätzliche Stichworte oder Hinweise einzugeben, die ebenfalls an das Modell übergeben werden. Ist hier Text eingetragen, überschreibt er die obigen Presets/Mood-Einstellungen komplett – man kann also gezielt eigene Beschreibungen vorgeben (z. B. „orchestral, melancholic, strings, 80s“), um die Tag-Generierung in eine bestimmte Richtung zu lenken.
 - **Export-Funktion:** Nach erfolgreicher Verarbeitung kann man über das Interface alle generierten Dateien gesammelt exportieren. Über ein Eingabefeld lässt sich ein Zielordner wählen, und per „Export“-Button kopiert das Tool alle relevanten Dateien dorthin (d. h. die Audiodateien sowie die jeweiligen `_lyrics.txt` und `_prompt.txt`). So kann man z. B. einen kompletten Datensatz-Ordner erstellen, ohne die Originaldateien zu verändern.
7. **Batch-Skripte & Erweiterungen:** Für Fortgeschrittene liegt dem Projekt auch ein Konsolen-Einstiegspunkt bei (siehe Installation), sodass man das Tool skriptgesteuert aufrufen kann (`ace-data` Befehl). Zusätzlich existieren im Verzeichnis `tools/` einige Helferskripte, z. B. zum Audio-Konvertieren (`mp3_to_wav.py`, `Batch wav_to_mp3.bat`) oder experimentelle Ansätze zur Genre-Klassifizierung mit vortrainierten Modellen (`classify_music.py` nutzt ein HuggingFace-Modell zur Genre-Erkennung). Diese sind optional und nicht zwingend für die Hauptpipeline erforderlich. Die Kerndatenpipeline konzentriert sich auf die oben beschriebenen Schritte Lyrics → BPM → LLM-Tagging.

Installation & Setup

Folgend wird die Einrichtung der Entwicklungsumgebung und Installation des ACE-Step Data-Tools erläutert. Es wird vorausgesetzt, dass Python und git installiert sind. Wir empfehlen die Nutzung von Miniconda zur Verwaltung der Python-Umgebung.

1. **Repository beziehen:** Klonen Sie dieses GitHub-Repository auf Ihren Rechner:

```
git clone https://github.com/methmx83/ace-data_tool.git
```

(Alternativ können Sie das Projekt auch als Zip herunterladen.)

2. **Python-Version & Umgebung:** Stellen Sie sicher, dass eine aktuelle Python-Version installiert ist. Das Tool erfordert Python ≥ 3.7 ; empfohlen ist Python 3.13 (getestet auf 3.13 in der Zielumgebung). Erzeugen Sie eine neue Conda-Umgebung (optional, aber empfohlen) mit der passenden Python-Version:

```
conda create -n ace-data_env python=3.13
conda activate ace-data_env
```

Dies stellt sicher, dass Abhängigkeiten isoliert und kompatibel installiert werden können.

3. **Abhängigkeiten installieren:** Wechseln Sie ins Projektverzeichnis und installieren Sie die benötigten Python-Pakete. Alle Abhängigkeiten sind in `setup.py` definiert – Sie können das Tool direkt als Paket installieren:

```
cd ace-data_tool
pip install -e .
```

Dies wird alle erforderlichen Bibliotheken (siehe unten) installieren und das Konsolenskript `ace-data` registrieren. Alternativ können Sie die Pakete auch manuell via `pip install -r requirements.txt` installieren, falls eine solche Liste bereitgestellt wird.

4. **Externe Anforderungen (Ollama & Modell):** Für die Tag-Generierung muss ein Ollama-Server mit dem benötigten Sprachmodell laufen. Installieren Sie daher Ollama (Version 0.9.6 oder neuer, siehe Ollama Doku) auf Ihrem System. Laden Sie anschließend ein kompatibles Sprachmodell in Ollama. Im standardmäßigen Config-File (`config/config.json`) ist der Modellname `"deep-xl_q4:latest"` eingetragen – dies deutet auf ein lokales Modell hin (z. B. ein quantisiertes Llama-Derivat). Stellen Sie sicher, dass entweder dieses Modell verfügbar ist oder passen Sie den `model_name` in der Config an ein installiertes Modell an. Starten Sie den Ollama-Dienst, damit die API unter `http://localhost:11434` erreichbar ist (Standardport von Ollama).
 - **Hinweis:** Ollama ist plattformübergreifend verfügbar (Windows erfordert evtl. WSL oder eine native Portierung, Version 0.9.6 unterstützt Windows nativ). Ohne laufenden LLM-Server kann die Tag-Generierung nicht erfolgen – in diesem Fall würde das Tool bei jedem Song nach den Lyrics stoppen.
5. **Ordner vorbereiten:** Legen Sie einen Ordner `data` im Projektverzeichnis an (falls nicht bereits vorhanden) und kopieren Sie Ihre Audiodateien dort hinein. Sie können auch die Unterordner-Struktur beibehalten, falls die Songs schon nach Alben/Genres etc. sortiert sind – der Scan läuft rekursiv.
6. **Starten des Tools:** Nach erfolgreicher Installation der Pakete und Einrichtung des Ollama-Modells können Sie die Anwendung starten. Unter Windows wurde bei Installation via `pip` das Skript `ace-data` registriert. Führen Sie in Ihrem aktivierten Env aus:

```
ace-data
```

Dadurch wird das Gradio-Webinterface hochgefahren. Nach einigen Sekunden sollte in der Konsole die Ausgabe `Running on local URL: http://127.0.0.1:7860` erscheinen. Öffnen Sie diese Adresse im Browser, um zur Benutzeroberfläche zu gelangen.

- (Sollte der Befehl nicht gefunden werden, prüfen Sie die Installation oder nutzen Sie alternativ `python -m webui.app` im Projektordner.)
7. **Nutzung über die UI:** In der Web-Oberfläche können Sie nun die gewünschten Einstellungen vornehmen (siehe oben: Overwrite Lyrics, Preset, Mood, Prompt) und dann „Start Tagging“ klicken. Das Tool verarbeitet daraufhin alle Dateien im `data/`-Ordner. Überwachen Sie die Log-Ausgaben, um etwaige Probleme (fehlende Lyrics, Modell-Fehler etc.) zu erkennen. Bei erfolgreichem Durchlauf finden Sie anschließend für jeden Track im `data/`-Verzeichnis zwei neue Dateien: `<Name>_lyrics.txt` und `<Name>_prompt.txt`. Nutzen Sie optional die Export-Funktion im UI, um alle Ergebnisse gesammelt in einen anderen Ordner zu kopieren (praktisch, um die Audios plus Tags z.B. in einen Trainingsdatensatz zu überführen).

Abhängigkeiten

Das Projekt verwendet aktuelle Python-Bibliotheken, um Audioverarbeitung, Web-Scraping und ML-Integration umzusetzen. Eine gekürzte Übersicht der wichtigsten Abhängigkeiten und Tools:

- **Python ≥ 3.10** (empfohlen 3.13) – neuere Versionen werden empfohlen, um Kompatibilität mit allen Libraries sicherzustellen.
- **Gradio (v5.35 oder höher)** – für die WebUI. Ermöglicht einfache Erstellung interaktiver Oberflächen zur Steuerung des Workflows.
- **Librosa** – für Audioanalyse (insb. Tempo/BPM-Erkennung). Intern nutzt Librosa `numpy`, `scipy`, `soundfile` etc., um Audiodaten zu laden und zu verarbeiten.
- **SoundFile** – zum Laden von Audiofiles (Librosa-Backend, erfordert die systemweite Bibliothek `libsndfile` – bei Installation über Conda wird diese automatisch mit installiert, bei `pip` ist sie im `pysoundfile` Paket enthalten).
- **Requests + BeautifulSoup4** – zum Herunterladen der Songtext-Webseiten und Parsen des HTML-Inhalts von Genius.
- **Tinytag** – zum Auslesen von Audiodatei-Metadaten (Künstler, Titel, Album etc. aus gängigen Formaten wie MP3, FLAC, M4A).

- **NLTK (Natural Language Toolkit)** – für Sentiment-Analyse und Text-Normalisierung der Lyrics. Das Tool lädt bei erstem Gebrauch die Sentiment-Daten (`vader_lexicon`) und Stopwortlisten herunter.
- **numpy, scipy, matplotlib** – übliche Pakete für numerische Berechnungen; `matplotlib` wird im Tool selbst kaum verwendet (evtl. war es für Debugging vorgesehen).
- **Ollama** – externer Dienst für LLM-Abfragen. Dies ist keine Python-Bibliothek, sondern eine Laufzeit, die lokal LLMs ausführen kann (ähnlich einem lokalen ChatGPT-Server). Es wird via HTTP angesprochen. Bitte installieren Sie Ollama separat, wie oben beschrieben. Alternativ könnten Sie theoretisch auch eigene API-Endpunkte eines anderen LLM (z. B. OpenAI API) nutzen, allerdings ist das Tool für das Ollama-Protokoll vorkonfiguriert.
- **CUDA 12 (optional, für GPU-Unterstützung im LLM-Betrieb)** – Wenn das verwendete Sprachmodell GPU-Beschleunigung nutzt (z. B. durch Ollama/llama.cpp mit CUDA-Unterstützung), sollten entsprechende NVIDIA-Treiber und CUDA-Toolkit installiert sein. Das empfohlene Setup nutzt CUDA 12.9 unter Windows für optimale Performance mit neueren NVIDIA-Karten.

Die Installation der Python-Abhängigkeiten erfolgt, wie oben gezeigt, über `pip/Conda`. Achten Sie darauf, in der passenden Umgebung zu installieren. Für Windows-Benutzer ist es empfehlenswert, Visual Studio 2022 Build Tools zu installieren (wie in der empfohlenen Konfiguration vorgesehen), da manche Python-Pakete (falls kein Wheel verfügbar) einen C++-Compiler benötigen. In unserem Fall werden aber die meisten Libraries als vorkompilierte Wheels installiert, sodass kein manuelles Kompilieren nötig sein sollte.

Hardware-Empfehlung: Das Tool wurde unter Windows 10 Pro x64 mit einem Intel i9 (24 Threads) und einer NVIDIA RTX 4070 (12 GB VRAM) getestet. 64 GB RAM standen zur Verfügung. Ähnliche oder etwas schwächere Konfigurationen sollten funktionieren. Mindestens 8–12 GB Grafikspeicher sind ratsam, damit das LLM-Modell in den GPU-Speicher passt – siehe Abschnitt unten zu Modellkompatibilität.

Beispiele zur Nutzung

Im normalen Gebrauch ist kein zusätzliches Coding nötig – das Tool erledigt alles auf Knopfdruck über die UI. Dennoch seien hier ein paar Anwendungsbeispiele und typische Outputs gezeigt, um das Verhalten zu illustrieren:

1. **Basic Run (alle Dateien verarbeiten):** Nachdem Sie Ihre MP3s in `data/` gelegt haben, starten Sie `ace-data`. Im Browser-UI können Sie z.B. das Genre-Preset auf “HipHop” setzen, den Mood-Slider auf leicht positiv (+0.5) und dann auf Start Tagging klicken. Angenommen, im Ordner befindet sich eine Datei `Song1.mp3` (Artist: Imagine Dragons, Title: Believer).
 - Das Tool wird: den Songtext laden, BPM berechnen (z. B. ~125), und Tags generieren. In der Log-Anzeige erscheint etwa:

```
Song1.mp3 - ✓ Saved lyrics and prompt
BPM: 125 - TAGS: bpm-125, male-vocal, rock, drums, energetic,...
```

Dies signalisiert, dass die Dateien erfolgreich erstellt wurden. Im Dateisystem finden Sie nun:

- `Song1_lyrics.txt` – den vollständigen Liedtext zu “Believer”, beginnend mit z.B. “Künstler: Imagine Dragons\nTitel: Believer\n\nFirst things first...”.
- `Song1_prompt.txt` – die Tags, z.B.:

```
bpm-125, male-vocal, rock, guitars, aggressive, anthem, pop-rock, upbeat,
```

(Dies ist ein fiktives Beispiel von ~10 Tags; die tatsächliche Liste kann leicht variieren. Wichtig ist das Format: klein, komma-getrennt, ggf. Bindestriche statt Leerzeichen.)

2. **Overwrite/Retry Lyrics:** Wenn ein Song beim ersten Lauf keine Lyrics gefunden hatte (im Log erscheint etwa “X Keine Lyrics gefunden.”), können Sie manuell korrigieren (z. B. ID3-Tags des Files verbessern oder den Songtext selbst als `<Dateiname>_lyrics.txt` im Ordner ablegen) und dann das Tool erneut ausführen. Aktivieren Sie in diesem Fall die Option “Overwrite lyrics”, damit das Tool trotz vorhandener Lyrics-Datei nochmal versucht, den Text online abzurufen (oder Ihre manuell erstellte Datei überschreibt). Ohne diese Option würde es vorhandene `_lyrics.txt` beibehalten.

3. **Nutzung als Python-Modul:** Alle Kernfunktionen sind auch programmatisch nutzbar. Beispielweise können Sie in einem eigenen Python-Skript auf die Module zugreifen, wenn Sie etwas automatisieren möchten:

```
from scripts.bpm import get_bpm
bpm_val = get_bpm("audio/path/song.wav")
print("Detected BPM:", bpm_val)

from scripts.lyrics import fetch_and_save_lyrics
fetch_and_save_lyrics("Imagine Dragons", "Believer", "output_folder/Believer_lyrics.txt")

from scripts.tagger import generate_tags, save_tags
tags = generate_tags("output_folder/Believer.mp3", prompt_guidance="rock, energetic")
save_tags("output_folder/Believer.mp3", tags)
print("Generated Tags:", tags)
```

Dieser exemplarische Code würde den BPM eines Songs bestimmen, Lyrics von Genius laden und Tags generieren, ohne die Gradio-Oberfläche zu nutzen. Beachten Sie, dass `generate_tags` Zugriff auf den laufenden LLM-Server erfordert – stellen Sie also sicher, dass Ollama mit dem Modell läuft, sonst wird ein Timeout/Fehler auftreten. Die Funktion `save_tags` speichert die Tags in der Datei `<Song>_prompt.txt` und sorgt dafür, dass ggf. die Lyrics-Datei von technischen Artefakten bereinigt wird (Entfernung von Sonderzeichen/Platzhaltern aus dem Text).

4. **Exportierte Datensatz-Struktur:** Wenn Sie die Export-Funktion nutzen oder manuell den `data/` Ordner nach der Verarbeitung woanders hinkopieren, erhalten Sie einen Datensatz, der für Trainingszwecke direkt genutzt werden kann. Ein möglicher Ausschnitt der Ordnerstruktur nach Verarbeitung könnte so aussehen:

```
data/
├── Album1/
│   ├── Track01.mp3
│   ├── Track01_lyrics.txt
│   ├── Track01_prompt.txt
│   ├── Track02.mp3
│   ├── Track02_lyrics.txt
│   └── Track02_prompt.txt
└── Album2/
    ├── SongA.mp3
    ├── SongA_lyrics.txt
    └── SongA_prompt.txt
```

Jeder Song liegt nun mit zugehörigem Lyrics- und Prompt-Textfile vor. Diese Textdaten können z.B. ins Training eines ACE-Step Modells eingespeist werden (Lyrics als gewünschter Gesangstext, Tags als musikalischer Kontext/Style). Beachten Sie, dass das Tool keine eigentlichen Audio-Features extrahiert – es generiert „nur“ beschreibende Meta-Informationen.

Modellkompatibilität & Hardware-Anforderungen

Die Qualität und Geschwindigkeit der Tag-Generierung hängt stark vom eingesetzten Sprachmodell (LLM) ab. Das ACE-Step Data-Tool wurde bewusst so gebaut, dass es kein externes Cloud-API benötigt, sondern mit einem lokalen Modell arbeitet – so bleiben auch urheberrechtlich geschützte Songtexte auf dem eigenen Rechner. Standardmäßig ist in `config/config.json` der Modellname `deep-x1_q4` eingestellt. Dies sollte ein kompatibles Modell im Ollama-Format sein (z. B. ein Derivat eines LLaMA-2 Modells mit ~7–13B Parametern, quantisiert auf 4 Bit für geringeren VRAM-Verbrauch).

- **Kompatible Modelle:** In der Regel eignen sich Chat-optimierte Modelle (z. B. LLaMA2-Chat, GPT-J, GPT-4-X Alpaca o.Ä.), die Anweisungen folgen können. Wichtig ist, dass das Modell ausreichend Kontextlänge ($\geq 4k$ Tokens) und Kompetenz hat, um auf Basis kurzer Lyrics + Regeln sinnvolle Tags zu erzeugen. Kleinere Modelle ($<7B$ Parameter) könnten Mühe haben, konsistente Genre/Mood-Tags zu liefern. Im Test lieferte ein LLaMA2-basiertes Modell (~7B, 4-bit) zufriedenstellende Ergebnisse.

- Theoretisch kann jedes über Ollama verfügbare Modell verwendet werden – um es zu ändern, passen Sie `model_name` in der Config an. Achten Sie darauf, dass das Modell die im Prompt vorgegebenen Markdown-Formatierungen und Rollen (“system” und “user”) versteht (die meisten neueren Chat-LLMs tun dies).
- **Hardware & Performance:** Das vorgesehene Modellsetup benötigt eine potente GPU. Mit der empfohlenen NVIDIA RTX 4070 (12GB VRAM) konnte das in `config.json` genannte Modell zügig geladen und Abfragen innerhalb von wenigen Sekunden pro Song beantwortet werden. Sollten Sie eine GPU mit weniger Speicher verwenden (z. B. 8 GB), könnte das Laden fehlschlagen oder der Modell-Server automatisch auf CPU-Betrieb ausweichen.
 - **Warnung:** Im CPU-Modus dauert die Tag-Generierung deutlich länger (mehrere Minuten pro Song sind möglich, abhängig vom Modell).
 - Falls nur wenig GPU-Speicher vorhanden ist, erwägen Sie:
 - ein kleineres bzw. stärker komprimiertes Modell (z. B. 4-Bit-Quantisierung, 7B Parameter oder weniger),
 - oder nutzen Sie Ollamas Optionen wie `--gpu-offloading` / `--cpu-offload`, um Teile des Modells auszulagern.
 - Grundsätzlich gilt: ≥ 12 GB VRAM sind empfehlenswert, um flüssig arbeiten zu können. Eine Multi-Core-CPU beschleunigt nebenbei auch das Audio-Processing (*Librosa*) und das HTML-Parsen, was aber meist nicht der Engpass ist.
- **Einschränkungen:** Beachten Sie, dass die Qualität der automatisch generierten Tags nicht perfekt ist – sie hängen vom Modellwissen und den Lyrics ab. Das Tool priorisiert Rap/Hip-Hop-Tags, wenn es Rap-Begriffe erkennt (siehe `Moods.md` für die vordefinierten Listen an Genres, Moods, Instrumenten, Rap-Stilen). Bei sehr ungewöhnlichen oder instrumentalen Stücken kann das Modell danebenliegen. Zudem werden Songs ohne auffindbaren Liedtext derzeit nicht getaggt – d.h. rein instrumentale Tracks erhalten keine Ausgabe außer dem Hinweis, dass Lyrics fehlen. Diese Einschränkung ist bewusst, da die Tags stark von lyrischen Inhalten und Stimmungen abhängen. In zukünftigen Versionen könnte man hier Audio-Feature-basierte Tagging-Modelle einbinden (siehe `tools/classify_music.py` für Ansätze), aber Stand Juli 2025 konzentriert sich das Tool auf textbasierte Metadaten.

Projektstruktur

Um den Einstieg in den Code zu erleichtern, hier eine vereinfachte Übersicht der Repository-Struktur:

```

ace-data_tool/
├── webui/
│   └── app.py - Gradio App (UI-Layout, Start/Stop, Button-Callbacks)
├── scripts/ - Kernfunktionen zur Datenverarbeitung:
│   ├── lyrics.py - Lyrics-Scraping (Genius API Scraper)
│   ├── bpm.py - BPM-Detection mit Librosa
│   ├── moods.py - Textanalyse (Sentiment, Tag-Bereinigung, Preset-Lader)
│   └── tagger.py - Tag-Generator (LLM-API Aufruf, Prompt-Definition)
├── include/ - Hilfsfunktionen und Konfiguration:
│   ├── preset_loader.py - Lädt Genre-Presets aus dem Ordner presets/
│   ├── clean_lyrics.py - Bereinigt Lyrics (Entfernt Sonderzeichen, Notizen)
│   ├── metadata.py, write_metadata.py - Alternativer Metadaten-Handler (JSON Export)
│   └── prompt_editor.py - (optional, UI-Element für Prompt-Bearbeitung)
├── presets/ - Vordefinierte Preset-Dateien (Genres, Stimmungen etc., vom Nutzer erweiterk
├── tools/ - Zusätzliche Tools/Optionals:
│   ├── convert_tools/ - Audio-Konvertierung (mp3 <-> wav Batch-Skripte)
│   └── wave_processing/ - Audioanalyse (z.B. Genre-Klassifikation mit ML, experimentell)
├── config/
│   └── config.json - Konfigurationsdatei (Pfad zum Input-Ordner, LLM-Modellname, API-URL,
├── README.md - (Diese Dokumentation)
└── System_Spezifikationen_music-scraper.md - Empfohlene Systemumgebung & Hardwaredetails
  
```

Jeder Bereich des Codes ist in Module gekapselt. Die UI-Schicht (`webui/app.py`) ruft Funktionen aus den Scripts auf und vermittelt zwischen Benutzeroptionen und Verarbeitung. In den Scripts liegen die Kernlogiken für jeden Verarbeitungsschritt. Besonders `tagger.py` ist komplexer, da hier die Kommunikation mit dem LLM erfolgt und ein mehrstufiges Fehlerhandling implementiert ist (inkl. Reload/Reset des Modells).

bei Problemen, mehrfache Versuche bis zu einer `retry_count`-Grenze). Die Include-Utilities kümmern sich um Datenvor- und -nachbereitung, während Presets und Config die anpassbaren Teile darstellen, die Sie ohne Code-Änderung modifizieren können (z.B. eigene Presets hinzufügen oder ein anderes Modell in `config.json` eintragen).

Schlusswort

Mit ACE-Step Data-Tool können Sie in wenigen Schritten aus einer Sammlung von Songs einen reich annotierten Datensatz erstellen. Das Tool nimmt Ihnen das mühsame manuelle Zusammensuchen von Liedtexten und das Erstellen von Tags ab und standardisiert das Format der Metadaten für weitere Verwendung im ACE-Step-Projekt oder ähnlichen Vorhaben. Bei Fragen oder Problemen konsultieren Sie bitte die Issues in diesem Repository oder wenden Sie sich an die Entwickler.

Tipp: Ausführliche Details zur getesteten Systemkonfiguration (Hardware, Treiber, Softwarestände) finden Sie in der Datei `System_Spezifikationen_music-scraper.md`. Diese enthält die exakten Umgebungsinformationen des Entwicklungsrechners und kann als Referenz dienen, falls Sie Umgebungsprobleme debuggen müssen. Viel Erfolg beim Einsatz des ACE-Step Data-Tools!