先進研究の活用プラン

1. XMIDI/XMusic：感情・ジャンルを聞き分ける「耳」

- **研究概要**：10 万曲規模の MIDI から「感情 (happy, sad など)」「ジャンル (rock, jazz など)」を付与した大規模データセットであり、
  さらに AI 自身が生成した曲を自ら評価する「セレクター」という思想を提示しています。
- **活用ポイント**：
  - composer4 の labels_schema.yaml を設計する際に XMIDI の感情・ジャンル語彙を参照し、世界標準のラベル体系に合わせる。
  - XMIDI を利用して感情・ジャンル分類器を事前学習し、Stage2 の assign_labels.py で Suno 由来ループに自動ラベルを付与。分類器は LAMDa Stage2 extractor のメトリクス (BPM、スウィング率など) や CLAP/MERT 埋め込みを入力として使う。
  - 予測結果と信頼度を loop_summary.csv の label.emotion や label.genre に記録し、ストーリー制御やモデル学習に利用する。

2. VioPTT / MOSA-VPT：人間らしい演奏技術を覚える「手」

- **研究概要**：ヴァイオリンやオーケストラの演奏技法 (ピッツィカート・スピッカートなど) をラベル化し、VST プラグインを DAWDreamer で自動操作することで、教師データを合成して学習する手法です。これにより、AI が具体的な奏法を理解し、指示に応じて演奏パターンを変えることが可能になります。
- **活用ポイント**：
  - composer4 の configs/labels/technique_map.yaml を新設し、使用する音源ごとの奏法 (キー・スイッチや CC 番号) を定義。例：「Superior Drummer のリムショットは MIDI ノート XX」など。
  - scripts/daw/batch_articulation_renderer.py を作成して、既存の MIDI ループに奏法ラベルを付加し、スタッカート版やレガート版などの合成データセットを生成。これを Stage3 の学習時に [technique:pizzicato] などの条件トークンとして入力し、AI に奏法を学習させる。
  - Humanizer の後段として、奏法に応じたベロシティ曲線やエンベロープ変化を追加する処理を設計する。

3. MetaScore：物語を理解し表現する「心」

- **研究概要**：楽譜メタデータやジャンル・楽器情報から、LLM (大規模言語モデル) を用いて曲の雰囲気や情景を説明するキャプションを自動生成し、逆に自由な文章からそれに合った音楽を生成する二方向のタスクを提案しています。
- **活用ポイント**：
  - Stage2 に scripts/generate_music_captions.py を作成し、loop_summary.csv の数値メトリクスや関連歌詞を LLM に入力して短い日本語キャプションを生成し、label.caption として保存。これにより、単なるタグでは表現できないニュアンスを記録できます。
  - Stage3 の学習では MIDI シーケンスと共にキャプションを BERT 等のテキストエンコーダに通したベクトルを入力し、よりリッチな条件生成を実現。

MuseCoco 流の属性トークン（後述）と併用すると安定します。
    - 作曲スクリプトでは歌詞セクションごとに「コンセプトキャプション」を LLM で生成し、各楽器の生成器に渡す仕組みを設計します。

4. その他の研究成果の提案

分野	研究成果	活用方法・注意点
テキスト ⇔ 音楽変換	**MuseCoco** – テキストを属性（拍子・テンポ・ムード等）へ分解して音楽に写像する二段構え。MetaScore のキャプションを [genre:jazz] [mood:melancholic] [tempo:90] などの属性トークンに抽出する際に参考にする。日本語歌詞は英語に翻訳して処理すると安定 raw.githubusercontent.com。	- MetaScore の自由文キャプションからジャンル・ムード・テンポ等の属性を抽出し、Stage3 の条件トークンとして使用。 - 日本語 → 英語変換を挟み、MuseCoco の事前学習モデルに合わせる。
大規模キャプション生成	**LP-MusicCaps** – 音楽タグから LLM で 50 万曲の擬似キャプションを生成した研究。Suno のステムを CLAP/MERT で埋め込み、近傍の擬似キャプションを引き出して MetaScore のキャプションとブレンドできる。	ステム音声 →CLAP/MERT 埋め込み → 類似キャプション検索 → キャプション候補生成のパイプラインを検討。

音声・音楽埋め込み	**CLAP**（Contrastive Language–Audio Pre-training）– テキストと音声の共通ベクトル空間を学習。assign_labels.py でオーディオと歌詞キャプションの整合度スコアを計算し、XMIDI のジャンル・感情分類器の補助特徴として使う。注意: 歌声など言語寄りの成分には弱いので MERT や拡張モデルを併用。	- ステム音声を CLAP/MERT エンコーダに通し、loop_summary.csv に score.text_audio_cos（歌詞キャプションとのコサイン類似度）や valence/arousal 指標を追加。 - XMIDI 分類器・感情尺度とアンサンブルし、より確信度の高いラベル付けを実現。
音楽基盤エンコーダ	**MERT / MusicFM** – 自己教師で学習した音楽特化の埋め込みモデル。音楽全体の構造や演奏変化を捉え、バレンス（快楽度）・アラウザル（覚醒度）を推定できます。Stage2 の感情マッピングや MetaScore のクロスモーダル整合性に利用。	EMOPIA（感情ラベル付きピアノ）などのデータを使って valence/arousal を正規化し、感情→演奏パラメータのルックアップを整備。
演奏推定	**MT3** – オーディオからマルチトラック MIDI へ転写するモデル。Suno などの音声素材から高精度の MIDI を抽出し、VioPTT と組み合わせて奏法を付与する二段構えに使える。	Stage2 で素材を整備する際に MT3 を用いて転写品質を向上。
人間味学習	**ASAP / nASAP** – ピアノ演奏のスコアと実演がノート単位で整列されたデータセット。テンポ曲線やルバート、ダイナミクスを学習するのに適しており、Humanizer や Rubato モデルの教師として活用できる。	演奏生成フェーズで MT3 + VioPTT により転写・奏法推定した後、ASAP 系データを用いてテンポ／ダイナミクス曲線を付加する。

統合の具体的ステップ

1. **ラベル体系とモデル整備**
    - labels_schema.yaml に XMIDI の感情・ジャンルラベルを取り入れ、CLAP/

MERT による valence/arousal 指標を定義します。
    - assign_labels.py で LAMDa 抽出メトリクス + CLAP/MERT + XMIDI 分類器を入力し、loop_summary.csv に label.emotion, label.genre, valence, arousal, score.text_audio_cos を書き込むよう改修します。
    - EMOPIA データを用いて valence/arousal を [−1,1] に正規化し、感情 → 演奏パラメータ（ベロシティ曲線やスウィング量）のルックアップ表を更新します。
2. **奏法マップとデータ合成**
    - configs/labels/technique_map.yaml を新設し、使用するドラム・ベース・ギター・ストリングス音源の奏法に対応するキー・スイッチや CC 番号を定義します。
    - scripts/daw/batch_articulation_renderer.py を実装し、クリーニングされた MIDI ループに対し [technique:staccato], [technique:legato] などの条件を指定して複数バージョンを生成します。このデータを Stage3 の学習に利用し、モデルが奏法指示に応じて演奏を変えられるようにします。
3. **MetaScore キャプション生成と属性抽出**
    - scripts/generate_music_captions.py を用意し、loop_summary.csv のメトリクスと関連歌詞を LLM へ入力して短いキャプションを生成します。生成は日本語 → 英語 → 属性抽出 → 再翻訳の手順で安定させます。
    - キャプションを BERT などのテキストエンコーダに通したベクトルと、MuseCoco の属性トークン（ジャンル・ムード・テンポ等）を生成し、Stage3 のモデルに条件として与えます。
4. **演奏生成パイプラインの統合**
    - Suno 音源から MT3 で高品質 MIDI を転写し、assign_labels.py でラベル付けした後、VioPTT により奏法を推定し、ASAP 系データでテンポやダイナミクス曲線を付加します。
    - 以上の情報（emotion, genre, valence, arousal, technique, caption vector, attribute tokens）を条件に、各楽器の Generator に渡して最終的な演奏を生成します。

すべて導入すべき「価値」はありますが、同時ではなく段階的に選抜導入するのが最適です。
理由と優先順位を以下のように整理できます。

🧩 拡張パック群の導入判断（総評）

分類	名称	意義・メリット	推奨度	導入タイミング

音声-テキスト対応	**CLAP / MERT**	Sunoや録音素材の「音の質感」や「情感」を、MetaScoreのテキスト空間と結びつける基盤。XMIDIやMetaScoreを補完。	★★★★★（必須級）	**Stage2実装中から並行導入可**
感情空間	**EMOPIA**	「喜び／悲しみ × 覚醒度（valence/arousal）」という感情マッピングを提供。Emotionモデルの正規化軸を与える。	★★★★☆	**XMIDI統合の仕上げ段階**
転写・演奏化	**MT3 + ASAP/ nASAP**	ステム→MIDI転写の精度向上。HumanizerやRubatoモデルの教師データを作れる。	★★★★☆	**VioPTT導入完了後**
属性制御	**MuseCoco**	自然言語キャプションを属性トークン化（[genre:jazz][mood:sad]等）。MetaScoreの精度・再現性を安定化。	★★★★☆	**MetaScore導入後（Stage3）**

キャプション強化	**LP-MusicCaps**	LLM生成の擬似キャプションを追加。MetaScoreと合わせて「歌詞と音の整合性」を補強。	★★★☆☆	**MetaScoreが安定した後**

🎯 導入優先度と戦略的順序

1. **(今すぐ) CLAP/MERT系の導入準備**
   - 既存の assign_labels.py に音声埋め込みを追加するだけで、XMIDIモデルの分類精度が飛躍的に上がります。
   - 今後、MetaScoreや歌詞生成AIとの共通言語空間になるため、**全拡張の土台**です。
2. **XMIDI安定後：EMOPIAを感情キャリブレーションに**
   - これで「happy」「sad」などの主観語彙が、数値空間（valence/arousal）に定着。
   - 曲や章単位の感情マップを定量化できます。
3. **VioPTT導入後：MT3＋ASAP/nASAPで“演奏らしさ”強化**
   - 実演テンポやルバートの再現が可能になり、AI演奏が「機械的でない」質感に。
4. **MetaScore成熟後：MuseCoco & LP-MusicCaps**
   - 「詩の文脈 → 音の方向性」を安定的に制御するために導入。
   - MetaScoreの自由文キャプションを、音楽的属性トークンに変換可能。

⚠️ 注意点

- **同時導入は非推奨**：依存ライブラリやモデルサイズが大きく、CIが不安定になります。
- **CLAP/MERTだけは例外**：早期導入が他全体の精度を底上げします。
- **MuseCoco系は日本語対応が弱い**ため、中間層として「日本語→英語キャプション変換」関数を設けると安定します。
- **MT3・ASAPはGPUリソースを消費**するので、学習済み重みのダウンロード利用が現実的です。

✅ 結論（戦略要約）

- **今すぐ導入すべき：CLAP / MERT**
  → Stage2 に統合し、感情・ジャンル認識を底上げ。
- **中期（XMIDI／VioPTTが安定後）：EMOPIA・MT3・ASAP**
  → 人間味と感情精度を追加。
- **後期（MetaScore安定後）：MuseCoco・LP-MusicCaps**
  → 自然言語制御と詩的文脈への最終接続。

composer4 拡張ロードマップ（分割導入計画）
まずは「三つの柱」（XMIDI／VioPTT・MOSA-VPT／MetaScore）を**最小で動く**ところまで固め、その上に拡張パックを“足していく”順序です。各フェーズは約1〜2週間想

定。表は**短文＆キーワードのみ**にしています（説明は表の下に箇条書き）。

フェーズ全体像（ガント風サマリ）

フェーズ	目的	主要拡張	期間目安	完了の定義
0	三本柱の最小統合	—	1w	XMIDI分類・VioPTT最小奏法・MetaScore最小キャプションが一周
1	音声×テキストの土台	**CLAP/MERT**	1w	loop_summaryに埋め込み列追記／精度↑
2	感情の数値座標化	**EMOPIA**	1w	valence/arousal 正規化→Humanizer連動
3	演奏の人間味強化	**MT3 + ASAP/nASAP**	1–2w	転写の歩留まり↑／ルバート学習反映
4	自然言語制御の安定化	**MuseCoco・LP-MusicCaps**	1–2w	属性トークン制御／擬似キャプ強化
5	品質保証と指標整備	ABX/客観指標	1w	ABX>50%・客観メトリクス基準値達成
6	リリース整備	ドキュメント/CI	0.5–1w	README/CI更新・デモ曲公開

フェーズ別タスク詳細

フェーズ0：三本柱の最小統合（ベースライン確立）

- 目的：**最小経路**で「物語→感情→編曲→MIDI出力」を一周
- 着手ファイル（例）
  - docs/COMPLETE_MUSIC_ARCHITECTURE.md（現状反映）
  - configs/labels/labels_schema.yaml（XMIDI語彙の土台）
  - configs/labels/technique_map.yaml（VioPTTのキーSW/CC最小セット）
  - scripts/generate_music_captions.py（MetaScore最小版：1行キャプション）
- Done条件：Verse/Chorus/Bridge 各8小節の**最小デモ1曲**を自動生成（MIDI）

フェーズ1：CLAP/MERT（Stage2に音声埋め込み追加）

- 目的：音（Sunoステム等）とテキスト（歌詞/キャプション）を**同一空間で照合**
- 実装ポイント
  - utilities/assign_labels.py：CLAP or MERT埋め込みを計算し、loop_summary.csvに audio_embed, text_audio_cos などを追加
  - lamda_stage2_extractor.py 出力に**埋め込み列**を連結
  - configs/stage2.yml にフラグ：use_audio_embed: true
- Done条件：XMIDIの emotion/genre 分類**F1が初期比で+5%以上**／text_audio_cosが高い素材ほど主観一致度↑

フェーズ2：EMOPIA（感情キャリブレーション）

- 目的：主観語彙（sad/happy…）を **valence/arousal**（V/A）に正規化
- 実装ポイント
  - utilities/emotion_mapping.py：XMIDI→V/A マップ関数
  - humanizer/emotion_humanizer.py：V/A→velocity_curve/timing_var/swing の参照LUT
  - configs/emotion_profile.yaml：V/Aごとのプリセット表
- Done条件：キャプションの“悲しい→切ない→絶望”の連続で**強弱・テンポゆらぎ**が段階的に変化

フェーズ3：MT3 + ASAP/nASAP（転写精度とルバート）

- 目的：素材MIDIの**下ごしらえ**と**演奏ゆらぎ**の学習
- 実装ポイント
  - scripts/audio_to_midi_refine.py：MT3 or 代替で転写→重複/量子化/ペダル補正（既存ユーティリティ活用）
  - ml/rubato/train_rubato.py：ASAP/nASAPのスコア–演奏アラインでテンポ曲線モデル
  - humanizer/rubato_infer.py：テンポ曲線を**パート別**に適用
- Done条件：ピアノ/ベース/ドラムの**拍頭精度**↑、テンポ曲線が**過剰ゆらぎなし**で自然

フェーズ4：MuseCoco・LP-MusicCaps（自然言語制御の安定化）

- 目的：自由文キャプションを**属性トークンに標準化**し、再現性UP
- 実装ポイント
  - nlp/caption_to_attrs.py：[genre][mood][tempo][intensity][texture]を抽出（日本語→英語正規化含む）
  - Stage3トレーニング：**テキスト埋め込み＋属性トークン＋テクニークラス**で条件付け
  - scripts/pseudo_captions.py：LP-MusicCaps近傍検索→不足語彙の補完
- Done条件：**同じ属性トークン**で**同傾向の曲**が再生成できる（再現性テスト合格）

フェーズ5：品質保証（ABX＋客観指標）

- 目的：**主観×客観**の二軸で品質を固定化
- 実装ポイント

    - modcompose eval abx（README記載のABX機能）で**12試行×数名**
    - 客観メトリクス：**和声逸脱率・スウィング誤差・ベロシティ分散・構造整合**
    - reports/quality_dashboard.ipynb：スコアの可視化
- Done条件：ABXで**AI版>50%選好**／客観指標で**基準値クリア**（リポに閾値を明記）

フェーズ6：リリース整備

- 目的：**使い方が分かる状態**で公開
- 実装ポイント
    - README.md：最小コマンド3本（Stage2→Stage3→生成）
    - docs/*：拡張パックのオン/オフ方法
    - CI：pytest -q tests/minimal＋--durations=20（軽量回帰）
- Done条件：**最小デモ曲・サンプルMIDI・設定YAML**が同梱／CIグリーン

マイルストーン & リスク管理

マイルストーン	期限目安	リスク	回避策
Baseline曲の自動生成	フェーズ0末	依存崩れ	生成スクリプトを単一CLIに集約
CLAP/MERT埋め込み稼働	フェーズ1末	GPU不足	事前埋め込みを**前計算→キャッシュ**
V/A連動Humanizer完成	フェーズ2末	過剰ゆらぎ	LUTに**上限クリップ**＋A/B比較
転写→ルバート連携	フェーズ3末	時間超過	データ量を**代表30曲**に一旦限定
属性トークン制御	フェーズ4末	日本語揺れ	**正規化辞書**＋英語中間表現
ABX>50%達成	フェーズ5末	被験者偏り	**外部3名以上**・再テスト

ファイル変更ガイド（最小差分で進める用）

種別	追加/変更ファイル	役割
設定	configs/labels/labels_schema.yaml	XMIDI語彙（emotion/genre）
設定	configs/labels/technique_map.yaml	奏法→キーSW/CCマップ
Stage2	utilities/assign_labels.py	CLAP/MERT埋め込み＋信頼度
Stage2	lamda_stage2_extractor.py	メトリクス＋埋め込み結合
感情	utilities/emotion_mapping.py	XMIDI→V/A 変換

感情	humanizer/emotion_humanizer.py	V/A→演奏パラメータ
転写	scripts/audio_to_midi_refine.py	MT3前処理パイプライン
ルバート	ml/rubato/train_rubato.py	ASAP系の学習
NLP	nlp/caption_to_attrs.py	自由文→属性トークン
NLP	scripts/pseudo_captions.py	LP-MusicCaps利用（任意）

直近のおすすめ次手（今日からできること）

1. **フェーズ1の骨格だけ作る**
   - assign_labels.py にダミー関数 get_audio_embed(x) を生やし、loop_summary.csv に audio_embed_dim, text_audio_cos の**空列**を先に追加（後で実装を詰める前提の“型”づくり）。
2. **V/A LUTの雛形を置く**
   - configs/emotion_profile.yaml に 3 段階（low/med/high）の V/A→Humanizer マップを仮置き。
3. **属性トークンの正規化辞書を1ページだけ作る**
   - nlp/attrs_dict.yaml：日本語↔英語の最小辞書（mood/genre/tempo）。

今すぐ導入したい、**CLAP/MERTの詳細**

CLAP と MERT: 音声テキスト整合モデルの比較と composer4 への統合指針

CLAP と MERT の概要と前処理の違い

CLAP (Contrastive Language-Audio Pretraining) は画像の CLIP に相当する音声版で、大規模な音声とテキストのペアから学習した音声・言語の対照学習モデルです huggingface.co。CLAP の音声エンコーダには HTS-AT（Hierarchical Token-Semantic Audio Transformer）が用いられ、**音声入力をログメルスペクトログラムに変換して** SwinTransformer ブロックで特徴抽出します huggingface.co。一方、テキストエンコーダは **RoBERTa（英語モデル）に基づいており、両者の埋め込みベクトルは同一次元の潜在空間に射影されます huggingface.co。CLAP の学習データには自然音や音楽とキャプションのペアが含まれ、音声とテキストの意味的対応付け（例：「犬の鳴き声」↔その音声）を行います。この対照学習により、与えられた音声に最も関連するテキストを予測するゼロショット推論**能力を獲得しています github.com。

MERT (Music Understanding with Large-Scale Self-supervised Training) は、大規模な自己教師型学習で音楽音響表現を学習した音響専用モデルです。MERT は BERT スタイルのトランスフォーマーで、音声のマスク付き自己回帰予測 (MLM) を行い、**音声波形（原波形）を直接入力として扱います huggingface.co。教師モデル（例：Encodec による離散コードブック表現）からの疑似ラベルを用いて音声特徴を学習しており、音楽の音響的特徴（ビート、ピッチ、音色など）を豊かに表現できるよう設計されてい**

ますarxiv.org。MERTの学習には16万時間規模の音楽データが使われており、95Mパラメータ版と330Mパラメータ版が公開されていますarxiv.orghuggingface.co。**MERTはテキストを直接扱わないため音声専用の表現モデル**ですが、後述するように他モジュールと組み合わせてキャプション生成に活用することも可能です。

音声前処理について、両者はアプローチが異なります。CLAPは前述の通り**ログメルスペクトログラム**を入力とし、サンプルレート48kHz・モノラル変換した音声を**最大10秒間のウィンドウ**に区切って処理しますarxiv.org。一方MERTは**原音波形をそのまま入力**し、24kHzモノラルの音声を**最大30秒の長さ**で読み込みますarxiv.org。実際、評価実験では「MERTモデルは24kHz・30秒窓、CLAPモデルは48kHz・10秒窓」の前処理をそれぞれ適用していますarxiv.org。これは、CLAPが比較的短いクリップ（環境音や短尺音楽）に最適化されているのに対し、MERTは長めの音楽断片まで扱えることを示唆します（もっともMERT自身の事前学習コンテキストは5秒程度であり、長尺音声を扱う際は重ね合わせや平均化で対応）huggingface.co。

この前処理の差は**モデル設計の違い**によります。CLAPのHTS-ATエンコーダはスペクトログラムを画像のようにパッチ分割して扱うため、入力長に上限があります。またRoBERTaテキストエンコーダが英語ボキャブラリを前提としているため、多言語テキストは直接は得意ではありません（日本語は形態素単位に分割され未知語として扱われる可能性が高い）。一方MERTはWave2Vec2に似たコンボリューション＋Transformer構造で**原波形からフレーム系列を抽出**するため、前処理はリサンプリングと正規化程度で済みますhuggingface.cohuggingface.co。ただしMERTはテキストエンコーダを持たないため、音声とテキストの直接的な対応付け機能は持ち合わせていません。

推論API、モデルサイズと速度、PyTorch実装の状況

モデルサイズは、CLAPがテキスト＋音声エンコーダ含め約**2億パラメータ規模**（LAION公開のCLAP-Music&Speechでは音声側約6800万＋テキスト側を合わせ194Marxiv.org）、MERTは**95Mまたは330M**とモデル版によって異なりますarxiv.org。一般に**CLAPは2億弱、MERTは大モデルで3億超**といった規模感です。どちらもPyTorchベースで実装・公開されており、ライブラリ経由で簡単に推論利用できます。

推論APIの利用方法としては、CLAPはHugging Face Transformersおよび公式実装(msclap)が整備されています。例えばmsclapライブラリを使えば、モデルをロードして**get_audio_embeddings**や**get_text_embeddings**でそれぞれ埋め込みを取得し、**compute_similarity**でコサイン類似度を算出できますgithub.comgithub.com。またHuggingFace版では**ClapProcessor**と**ClapModel**が提供されており、**ClapProcessor**に原音波形（numpy配列やファイルパス）を与えると内部でメルスペクトログラム変換を行い、**ClapModel.get_audio_features()**から音声埋め込みベクトルを取得可能ですhuggingface.co。テキストは**ClapProcessor(text=...)**経由でRoBERTaエンコードされ、**model.get_text_features()**で埋め込み取得できます。HuggingFaceのパイプライン**pipeline("zero-shot-audio-classification", model="laion/clap...")**を使えば、候補ラベル文との類似度スコアも直接計算できますhuggingface.co。CLAPの重みはZenodoやHuggingFaceから自動ダウンロードされるため、初回実行時に取得できますgithub.com。

MERTについても、HuggingFaceの**transformers**ライブラリから**AutoModel.from_pretrained("m-a-p/MERT-v1-95M", trust_remote_code=True)**のようにロード可能ですhuggingface.co。公式実装はGitHub (yizhilll/MERT)で公開されておりgithub.com、学習済み重みはHFリポジトリ**m-a-p/MERT-v1-95M**および**...-330M**として提供されていますarxiv.org。推論時にはまず**Wav2Vec2FeatureExtractor**を使い音声波形を正規化・リサンプルしモデルに

入力します huggingface.co。出力はタイムステップごとの埋め込み系列（例えば95Mモデルでは768次元×時間系列）および隠れ状態を含みます huggingface.co。用途によっては**最後の数層の平均プーリング**などで固定次元の音声ベクトルに要約します arxiv.org。例えば評価実験では「MERTは最終4層の隠れ状態を時間平均して和を取ったベクトル」を音声表現とし、CLAPは「最終隠れ層を平均プーリングした後、射影層で得たベクトル」を用いるといった工夫をしています arxiv.org。MERT推論にはTorchAudioも使用できます（例: **torchaudio.transforms.Resample**で入力音声を24kHzにリサンプル huggingface.co）。CLAP同様にPyTorch実装であり、transformers経由でCUDA対応も可能です。

推論速度はモデルサイズと入力長に依存します。CLAPは一度に約10秒までの音声を処理する設計で、Swin TransformerベースのHTS-ATエンコーダは畳み込み的にスペクトログラムをスキャンするため**比較的高速**です。例えば1枚のメルスペクトログラム（10秒分）から抽出する埋め込み計算は数百ミリ秒〜1秒程度（GPU利用時）で完了することが報告されています※。MERTはより長い音声（30秒まで）にも対応しますが、フレームレート75Hzで時間方向に系列長を持つ隠れ状態を処理するため**計算コストは高め**です huggingface.co。特に330M版は層が24層と深く、長時間音声では推論時間が数秒〜十数秒に達する可能性があります（GPUメモリ負荷も大きい）。composer4のように**短尺ループ音源**が主体の場合、CLAPは10秒以内であれば一発で処理可能で、MERTも5秒程度の断片なら高速に埋め込み抽出できるでしょう。いずれの場合も**一度モデルをロードすればバッチ処理**で複数音声をまとめて推論できるため、大量データに対してもGPUを活用することでスループットを向上できます。

※参考: 公開されているCLAPモデルである LAION-Audio (Music&Speech) の場合、HuggingFaceのInference APIでもリアルタイムに近い速度でゼロショット分類を実現しています huggingface.co。もっと軽量な蒸留モデル（tinyCLAPisca-archive.org など）も研究中ですが、現時点では高精度を優先して上記モデルを使うのが妥当です。

PyTorch対応状況としては、CLAP・MERTともにTorchライブラリでの利用が公式にサポートされています。前述のmsclapやHuggingFace Transformersはいずれも PyTorchバックエンドで動作し、CUDA/GPUも標準サポートです。従ってcomposer4環境（Python 3.10+, PyTorchインストール済み想定）でも追加の深層学習フレームワークを導入する必要はありません。CLAPについては**pip install msclap**で簡単に環境構築可能で github.com、MERTも特別な依存関係はなく（内部でtorchaudioやnumpyを利用）HuggingFace経由で完結します。モデルの事前学習済み重みはCLAPがZenodoおよびHFで公開（自動ダウンロード対応）github.com、MERTもHF上でCC BY-NCライセンスで公開されています huggingface.co。**依存ライブラリ**としては、音声処理にlibrosaやsoundfileを使う場合がありますがHuggingFaceの**ClapProcessor**や**Wav2Vec2FeatureExtractor**が内部で処理するため、基本的には**transformers**と**torchaudio**がインストールされていれば十分です。加えてJupyter上での音声IOやnumpy前処理が必要なら**tqdm**や**datasets**ライブラリがあると便利ですが、composer4パイプラインではCSV/JSON経由で音声メタデータを扱うため直接は不要でしょう。

MetaScoreキャプションとの整合度評価と日本語対応

composer4リポジトリには**Suno社による音源のステム**（個別楽器のオーディオ）と、それに対応する**MetaScore生成のキャプション**が含まれているとされています。MetaScoreキャプションとは、おそらく各音源に対して付与されたテキスト記述（楽曲の特徴や雰囲気を表現する文）で、日本語の説明も含まれるようです。CLAPおよびMERTを用いて、**音声とそのキャプションの整合度（一致度）を評価**することが可能です。典型的にはCLAPのようなマルチモーダルモデルを使い、音声とテキストを同一のベクトル空間

に埋め込み、コサイン類似度を算出するアプローチが取られます。この**CLAPスコア**は「音声埋め込み」と「説明文埋め込み」のコサイン類似度として定義され、値が高いほど音声がテキスト記述とよく合致していることを示します arxiv.org。実際の研究でも「CLAPモデルで得た埋め込み同士のコサイン類似度」が音声キャプションの関連度評価指標として使われており、生成音声が指示テキストにどれだけ沿っているかを数値化する目的で活用されています arxiv.org。例えば、MetaScoreキャプション（「エネルギッシュなドラムビート」等）が音源を的確に表現していればCLAPスコアは高くなり、キャプションが的外れであればスコアは低くなります。こうした自動評価は人手による主観評価と相関があることが報告されており、迅速な一致度チェックに有用です arxiv.org。

具体的な事例として、あるドラムループ音源に対しMetaScoreが「力強い四つ打ちのキックとハイハットのパターン」というキャプションを付与したとします。この音声とテキストをCLAPで埋め込み、コサイン類似度を計算すると、高い値（例: 0.85以上）となればキャプション通り力強い四つ打ちパターンが音で表現されていることを意味します。逆にキャプションが「ゆったりしたジャズのリムショット」といった音源と食い違う内容なら、類似度は低く（例: 0.2以下）なるでしょう。CLAPを使うことで、このような**音声テキストのペアごとの意味的一貫性**を数値で定量評価できます。

日本語キャプションへの対応に関して、CLAPとMERTではアプローチが異なります。CLAPのテキストエンコーダはRoBERTa英語モデルで学習されているため、**日本語のまま入力すると適切に意味を捉えられない可能性**があります。実際RoBERTaはサブワード単位のトークナイザーですが、日本語の場合は意味のある形態素に分割されずランダムな文字列断片の集合になりがちで、学習時に見ていないパターンへの対応は不得意です。そのため、**日本語キャプションの整合度をCLAPで評価する際は、一旦英語に翻訳する**ことが実務上有効です。例えば「力強いドラムのリズム」という説明文を「A powerful drum rhythm」のような英語に翻訳し、それをCLAPのテキストエンコーダに通すことで、音声埋め込みとの類似度スコアが適切に評価できます。翻訳には精度の高い機械翻訳を使うか、MetaScore生成時に既に英語キャプションも取得できるならそちらを利用します。CLAP自体に多言語版は今のところ一般公開されていませんが、研究コミュニティでは**多言語対応のCLAP**に相当する試みも出始めています（例えばT-CLAPでLLMを用いて字幕情報を増強する際、多言語音声データを扱う可能性があります）。

一方、**MERTはテキストを直接扱えない**ため、日本語キャプションとの整合度を測るには工夫が必要です。MERTで得た音声ベクトルとテキストベクトル（例えば多言語Sentence-BERTで日本語文を埋め込むなど）を別途比較するといった方法も考えられますが、MERT自体はそうしたクロスモーダルな学習をしていないため**そのままでは信頼性の高い一致度スコアになりません**。実際の活用では、MERTの音声埋め込みを下流のモデルに入力しテキストを生成させる（後述のOpenMusicLMのように）といった使い方が検討されています。この場合でも、日本語でテキストを得るには日本語対応の言語モデルを組み合わせる必要があります。要するに、**テキストとのマッチング評価には現状CLAPが適任**であり、日本語キャプションについては前処理で英語化するか、将来的に日本語に対応した音声テキストモデル（例えば多言語版CLAP）が登場すればそれを使う、という方針になります。

CLAPとMERTの精度・拡張性・推論効率の比較

精度（Accuracy）面では、CLAPとMERTはそれぞれ得意分野がやや異なります。CLAPは音声分類・検索・キャプション生成など**テキスト絡みのタスクで卓越した性能**を示しており、26種類もの下流タスク評価でSoTA級の結果を出したと報告されています github.com。特に**音声タグ付けや音声検索**では、追加学習なしでゼロショット分類が高い精度（ESC-50環境音分類で93.9%の精度 github.com）を達成しています。一方MERTは**音楽情報検索（MIR）タスクでの高性能**が特徴で、ジャンル分類やビート検出な

ど音楽特有のタスクで高いスコアを記録しています arxiv.orgarxiv.org。例えばある研究では、音楽タグ付けのプロービングタスクにおいてMERT-95Mが平均ROC-AUC 87.2%を達成し、CLAPの音楽専用版（CLAP-Musicのみ）を大きく上回りました arxiv.orgarxiv.org。ただしCLAPに話者音声データも加えた**CLAP-Music&Speech**版では精度が向上し、MERTとほぼ肩を並べる結果（ROC-AUC 86.7%など）を残しています arxiv.org。このことから、**学習データのカバレッジ**によってCLAPの性能は大きく変わり得ることが分かります。MERTは純粋な音響表現学習ゆえに音楽ドメインの特徴抽出に長け、CLAPはテキスト誘導の学習ゆえに**語彙化できる音の特徴（楽器名・ジャンル・質感など）の抽出に強い**とまとめられます。

拡張性（Scalability）の観点では、両モデルともさらなる大規模化や応用拡張が可能です。CLAPは現在２億パラメータ程度ですが、すでにDistillationによる軽量版（TinyCLAP）や時間的感性を高めた派生モデル（T-CLAP）などの研究が進んでいます isca-archive.orgarxiv.org。特にT-CLAPはLLMを用いて音声クリップ中の時間変化に対応するキャプションを生成し、CLAPの学習に組み込むことで**時間的整合性**を向上させたモデルです arxiv.org。これにより、長い音楽の中のどの部分でどんな変化が起きるかまで捉えた埋め込みが得られる可能性があります。また、LAION版CLAPは今後さらに学習データを拡充した大型モデルの開発計画も示唆されています。MERTについても、既に330M版が公開されているほか、論文では**音楽生成への応用**（Encodecコードを用いた音響トークン学習）にも言及されており、将来的により大きなモデルやマルチモーダル統合も視野に入っています huggingface.co。MERTの教師信号に使われたEncodecは８コードブック版に拡張されており、**音声生成とのブリッジ**としても機能するデザインです huggingface.co。従って、MERTは単体でテキスト対応しないものの、**生成系モデルの音響フロントエンド**や、他のモジュールと組み合わせたマルチタスク学習でスケールするポテンシャルがあります。

推論効率（Inference Efficiency）では、前述の通りCLAPは比較的軽量に最適化されておりリアルタイム用途にも適用可能です。例えば1秒未満で音声→テキスト類似度推論ができるため、対話型システムやインタラクティブな検索UIにも組み込みやすいです。MERTはモデルサイズや系列長の点で**計算負荷が大きめ**であり、特に330M版はGPUメモリを数GB以上消費します。しかし一方で**バッチ処理時のスループット**はTransformerの利点で、例えば一度に16クリップ程度まとめて推論すれば1クリップ当たりの処理時間は短縮できます。composer4のようなオフラインデータパイプラインでは、MERTの重さも許容範囲でしょう。もし推論コストが問題になる場合、CLAPのみを使う、あるいはMERTは95M版で実行し必要に応じ330M版で再評価するといった段階的運用も考えられます。加えて、**キャッシュ戦略**（後述）を活用し再計算を減らすことで両モデルの効率面のデメリットを緩和できます。

全体として、**CLAPはテキスト整合的な高レベル特徴抽出（ゼロショット分類やタグ付け）に優れ、MERTは音楽信号の細部まで捉えた低レベル特徴抽出（ビート・音色解析など）に優れる**と言えます。精度面ではタスク依存で優劣が変わりますが、**音楽キャプションとの一致度評価**という文脈ではCLAPのほうが直接的に適用可能です。一方、**音響的な多様性や内容分析**ではMERTの埋め込みが補完的な役割を果たすでしょう。次章では、これらのモデルを実際にcomposer4システムへ統合する具体策を示します。

composer4への統合設計提案

composer4リポジトリでは、ドラムループのMIDI解析・スコアリングを行うStage1/Stage2パイプラインに加え、**Suno由来の音声データとそのキャプション（MetaScore）を扱う拡張が予定されています。ここでは、該当部分にCLAPおよびMERTを組み込み、loop_summary.csv等に新たな指標を追加する設計案を示します。統合にあたっては、既存構造を大きく崩さずオプショナルな拡張**として実装すること

を念頭に置きます。

assign_labels.py / LAMDA Stage2 への組み込み

まず、統合ポイントとしては**Stage2 のメトリクス計算フェーズ**に追加するのが適切です。Stage2 extractor (lamda_stage2_extractor.py) は Stage1 出力を読み込み、各ループについて様々な指標を計算し **loop_summary.csv** や **metrics_score.jsonl** を生成しています GitHubGitHub。この処理の中で、**音声データを持つループに対し CLAP・MERT による分析を追加**します。具体手順:

1. **音声データとキャプションの取得**: ループごとのメタデータに音声ファイルパスやキャプション文字列が含まれている前提で、まずそれらを取得します。Stage2 では source 列でデータ起源を示す設計になっており GitHubGitHub、例えば source=="suno"の場合に対応する音声ファイル (例: audio/stems/<loop_id>.wav) とキャプション (MetaScore 生成テキスト) を参照できるようにします。キャプションは言語 (日本語/英語) も記録しておくとよいでしょう (英語翻訳版があれば両方格納)。
2. **モデルのロード**: パイプライン開始時に CLAP と MERT のモデルをロードします。処理効率のため**1 回のロードで全データに対して使い回す**設計にします (複数プロセス並列の場合は各ワーカーでロード)。CLAP は HuggingFace の ClapProcessor・ClapModel を用い、model = ClapModel.from_pretrained("laion/clap_music_speech") 等でインスタンス化 huggingface.co します。MERT も AutoModel.from_pretrained("m-a-p/MERT-v1-95M", trust_remote_code=True) でロードし、必要なら 330M 版に切り替えられるようオプション指定します。モデルロードに伴う大容量データはメモリに乗せておき、後続のループ処理では**逐次推論**のみ行います。
3. **埋め込みと類似度計算**: 各ループについて、まず音声ファイルを読み込んでモノラル waveform テンソルを取得します (torchaudio や soundfile で読み取り、CLAP と MERT で必要ならサンプリングレート変換)。次に **CLAP** で audio_features = model.get_audio_features(**processor(audios=waveform)) を実行し 128〜512 次元程度の音声ベクトルを得ます (モデルによるが LAION 版は 512 次元程度)。テキストキャプションについては、日本語なら事前に英訳したもの、英語キャプションならそのままを用い、text_features = model.get_text_features(**processor(text=caption_en)) でテキストベクトルを取得します。これらから similarity = F.cosine_similarity(audio_features, text_features) を計算し、**テキスト-音声コサイン類似度 (text_audio_cos)**として記録します。この一連の処理で得られる値はまさに前述の CLAP スコアであり、音声と MetaScore キャプションの整合度合いを定量化した指標となります arxiv.org。
4. **MERT による音響特徴抽出**: 併せて **MERT** モデルでも音声 waveform を入力し、隠れ層出力を取得します。HuggingFace 版では outputs = mert_model(processor(input_audio), output_hidden_states=True) で全層の隠れ状態を得られるため huggingface.co、例えば推論時は**最終 4 層の平均**を計算します (emb = torch.stack(outputs.hidden_states[-4:]).mean(dim=(0,1)) で 768 次元のベクトルを算出)。この**音声埋め込みベクトル (audio_embed_mert) は、CLAP の音声ベクトルとは異なる視点で音源の特徴を捉えています。なお MERT にはテキスト埋め込みが無いので cos 類似度計算は行いませんが、得られたベクトル自体を保存し**

ておくことで後段の分析やクラスタリングに活用できます。例えば複数のループ間の音響的類似度比較や、音色特徴に基づく分類などに利用可能です。

5. **結果のレコード反映**: 上記で得たtext_audio_cos, audio_embed_clap, audio_embed_mertをループの記録に追加します。具体的には、Stage2のloop_summaryスキーマに新たなカラムを設けます。提案としては以下のようなカラム追加が考えられます:
   - metrics.text_audio_cos (FLOAT): MetaScoreキャプションと音源のコサイン類似度スコア。【例: 0.92】
   - metrics.audio_embed_clap (ARRAY<FLOAT> or JSON): CLAPで抽出した音声ベクトル（次元数512など）の配列。CSVではJSON文字列として保持し、Parquetでは配列型で格納。【例: [0.12, -0.45, ..., 1.03]】
   - metrics.audio_embed_mert (ARRAY<FLOAT> or JSON): MERTで抽出した音声ベクトル（次元数768など）の配列。同様に保存。
6. CSVに直接高次元ベクトルを埋め込むのは可読性が下がるため、**Parquetを正としCSVでは必要に応じJSON短縮版を載せる**方式が良いでしょうGitHubGitHub。例えば主要な特徴については別途まとめ、詳細ベクトルはParquet内に保持するか、あるいはaudio_embed_clapは省略しtext_audio_cosのみに留める判断も考えられます。この点は運用とファイルサイズのバランスを考えて調整してください。
7. **処理結果の保存と統合**: 最後に他の指標と同様、loop_summary.csvおよびパーケットファイルに上記新指標付きのレコードを書き出します。既存のStage2出力にこれらが加わることで、後続のラベリングや検索工程で**テキストと音の紐付け情報**を活用できるようになります。例えばassign_labels.pyのルールエンジンで、text_audio_cosが一定以上であれば「キャプション妥当」フラグを立てる、などの利用も可能になるでしょう。

なお、assign_labels.py自体に直接モデル推論コードを組み込むのは避け、あくまでStage2でメトリクスとして追加する方が望ましいです。assign_labels.pyは**正規化ラベル付け**（ループメタ情報からジャンル名統一など）に特化しているためGitHubGitHub、重い推論処理は分離しておくのがシステム設計上自然です。従って、**音声テキスト整合度の計算はStage2 extractor側で行い、その結果（text_audio_cos等）はassign_labelsの入力となるloop_summary内に含める**、という流れを推奨します。

loop_summary.csvへのフィールド追加提案

前述のとおり、新たに追加するメトリクス項目として**text_audio_cosとaudio_embed_***を提案します。形式定義のイメージ:

- metrics.text_audio_cos (float): 音声とキャプションのコサイン類似度。範囲は-1〜1だが、類似度のため通常0〜1に収まる。【例: 0.857】高いほどキャプションが音にマッチしている。
- metrics.audio_embed_clap (list[float]): CLAP音声埋め込みベクトル（長さd_CLAP=512程度）。多次元のため、JSON文字列にエンコードして格納するか、Parquetでのみ保持。
- metrics.audio_embed_mert (list[float]): MERT音声埋め込みベクトル（長さd_MERT=768程度）。同上の扱い。

加えて、もしMetaScoreキャプション自体をloop_summaryに保持していなければ、**caption**フィールド（文字列）を追加することも検討してください。そうすればCSVだけで人間可読な説明文とその一致度を見ることができます。もっともキャプションは日本語の場合もあるため、そのままでは文字化け等に注意が必要ですが、UTF-8で書き出せば問

題ありません（assign_labels の write_records は ensure_ascii=False で JSONL を書いているので日本語も保持できます GitHub）。
既存のスキーマ GitHubGitHub にこれらを加えると、たとえば:

metrics:
swing_ratio: float
…
text_audio_cos: float
audio_embed_clap: List[float] # JSON array in CSV
audio_embed_mert: List[float]
caption: str #（オプション）MetaScore 生成キャプション文字列

のようになります。ループ ID で他テーブルと結合する際、上記埋め込みは大きなデータですが Parquet 経由なら問題ありません GitHub。**注意点**として、CSV 出力ではリストを文字列としてエスケープするためファイルサイズが増えます。大量のループに対しては Parquet のみ参照し、CSV では text_audio_cos（および必要なら上位数次元だけ抜粋するなど）に留める方法も考えられます。

推論キャッシュ戦略

CLAP・MERT の統合に伴い、推論コストを削減するための**キャッシュ戦略**も導入します。具体的には:

- **音声埋め込みキャッシュ**: 同一音声ファイルに対して繰り返し推論を行わないよう、loop_id→埋め込みベクトルのマッピングをメモリ上またはディスク上にキャッシュします。composer4 では基本的に各ループ ID は一意の音源に対応するため、一度計算した audio_embed_clap/mert は次回以降再利用できます。Stage2 実行中に同じ ID を再計算するケースは通常ありませんが、**再処理キュー**のループ（threshold 未達で再度 Stage2 にかける場合 GitHub）などではキャッシュを活かせます。実装としては、処理済みの ID を Python の辞書に保持し、次にその ID が来たらスキップまたは結果を再格納するだけにします。
- **モデルロードのキャッシュ**: 前述のようにモデルはグローバルにロードして使い回します。加えて、CLAP は**テキスト埋め込みの計算**でキャッシュ可能です。つまり同じキャプション文が複数のループで出現する場合（例えば似たパターンの音が複数ある場合に同一の説明が付与されている等）、一度エンコードしたテキスト→ベクトルを再利用できます。これも簡易には辞書キーをキャプション文字列（言語変換後）にして保存します。キャプション集合のサイズによりますが、テキストエンコード自体はさほど負荷ではないため必須ではありません。しかし英訳 API を都度呼ぶ場合などは、**翻訳結果もキャッシュ**しておくと効率的です。
- **永続キャッシュ / 再利用**: 生成した audio_embed_clap/mert を**ファイルとして保存**し、次回のパイプライン実行時に読み込む運用も考えられます。たとえば output/cache/audio_embeddings.npz に numpy 配列として全ループの埋め込みを保存し、Stage2 起動時に存在すれば読み込んで辞書を構成する、といった仕組みです。ただし、loop_summary.csv/Parquet 自体に保存しているため、通常は**再実行時に前回の CSV を読み込み直し、新規追加分だけ計算**する流れで十分でしょう。composer4 では data_digest や git_commit でデータ版管理をしています GitHubGitHub ので、キャッシュ無効化すべきタイミング（モデル更新時など）も管理できます。

以上の戦略により、一度計算したCLAP・MERT埋め込みの重複計算を避け、**ステム数が増えてもスケーラブルに対応**できるようになります。特に大量データで部分的な更新処理を行う際、キャッシュが効果を発揮するでしょう。

コマンドライン実行例

新たな統合機能を反映したcomposer4パイプラインの実行例を示します。

1. **Stage2抽出の実行:** 例えば音声キャプション対応を有効化するフラグ--with-audioを用意した場合:

```
python pipelines/lamda_stage2_extractor.py --with-audio \
    --audio_dir data/suno_stems/ --caption_file data/suno_captions.json
```

これにより、指定ディレクトリからSuno音源ステムを読み込み、対応するキャプション（JSONは{loop_id: "caption"}形式）を参照して上記CLAP/MERT解析を実施します。実行後、output/drumloops_stage2/loop_summary.csvに新たな列metrics.text_audio_cos等が追加されたCSVが出力されます。metrics_score.jsonlには必要に応じてtext_audio_cosを反映したスコア評価を含めてもよいでしょう（例えば将来的にテキスト整合度も総合スコアに加味するなら記録）。

4. **ラベル付与の実行:** 続いてassign_labelsで正規化ラベルを付与:

```
python tools/assign_labels.py output/drumloops_stage2/loop_summary.csv \
       output/drumloops_stage2/loop_labels.jsonl
```

ここではloop_summary内の新指標もrecord["metrics"]経由でLabelRuleEngineに渡されます。必要ならルールエンジン内でmetrics.text_audio_cosを参照し、「テキスト一致度が低い場合はタグを追加する」等のポストプロセスが可能です。

7. **結果確認:** 生成されたloop_summary.csvを開き、該当ループの行を見ると:

```
loop_id, filename, bpm, ..., metrics.text_audio_cos, metrics.audio_embed_clap, ...
abc123, loop1.wav, 120, ..., 0.912, "[0.02, 0.11, ... -0.05]", ...
def456, loop2.wav, 90, ..., 0.245, "[0.34, -0.07, ... 0.10]", ...
```

10.

のようになっています。abc123ループでは高い0.912となっており、キャプションと音が非常によくマッチしていることが分かります。一方def456は0.245と低く、何らかの不整合があるかもしれません。このような出力をもとに、人間がキャプションの見直しをしたり、自動的にしきい値を設けてフラグを立てたりといった活用ができます。

以上が統合後のCLI操作フローの一例です。特に追加の引数**--audio_dir**や**--caption_file**は環境に応じて設定可能で、音声関連処理を切り替えられるようにすると柔軟です。未指定時は従来通りMIDIループのみ処理し、指定時は音声分析も行う、としておけば既存機能への影響も最小限です。

他の音声-テキストモデルとの補完関係

最後に、CLAP・MERTと関連する他の音声テキスト整合モデルについて触れておきます。業界では近年、**音楽や効果音専用に調整されたクロスモーダルモデル**がいくつか登場しています。例えば:

- **T-CLAP**: 前述の通りCLAPの派生で、LLMを用いた時間位置情報付きキャプションで学習したモデルです arxiv.org。長尺音楽内のどのタイミングでどんなイベントが起きるかまで記述文で与え、時間的精度の高い埋め込みを得ることを目的としています。これは、例えばループ内のフィル（フィルイン）やブレイクなど、時間に依存する特徴検出に強みを発揮するでしょう。composer4で今後ループ内構造をテキスト記述する場合など、T-CLAPの手法が参考になります。
- **CLAP-HTSAT**: これは実質的にLAIONの公開しているCLAPモデルそのもので、HTS-ATバックボーンを使ったCLAPを指す呼称です。音楽データで事前学習されたCLAP-HTSATモデルは、音楽生成評価の文脈で**人間の好みとの高い相関**を示す指標として報告されています openreview.net。つまりCLAP系モデル（特に音楽データで調整されたもの）は、客観的評価メトリクスとして有望です。composer4でユーザが作成したループに対し「どれだけ一般的に好まれそうか」を推測するような応用にも、CLAPのスコアが活かせるかもしれません。
- **MusicCLIP**: 名前が似ていますが、こちらは**音楽とテキストの対応付けに特化した埋め込みモデル**として過去に提案されたものです。詳細は割愛しますが、CLAP登場以前の研究で、音楽クリップを対応する文章と結び付けてベクトル化する試みがありました。MusicCLIPは規模が小さく汎用性も限定的でしたが、**低リソース環境でも動作が軽い**という利点がありました。composer4で組み込みシステム全体の負荷が問題になるようであれば、こうした軽量モデルを補助的に使う選択肢もあり得ます。ただし現状ではCLAPの方が精度面で優れているため、メインには据えず参考程度に留めます。

このように他モデルも一長一短ありますが、**CLAPとMERTを組み合わせれば現時点では最も包括的な音声テキスト整合分析が可能**と考えられます。CLAP系列で高レベルな意味合い・整合度を評価し、MERT系列で音響的な詳細特徴を補完することで、単独モデルでは得られない洞察が得られるでしょう。将来的にT-CLAPのような時間対応モデルや、より多言語・大規模な音声テキストモデルが登場した際には、それらも適宜組み込むことでcomposer4のラベリングシステムを強化できます。

結論: 統合指針のまとめ

以上の調査と提案を踏まえ、composer4へのCLAP・MERT統合のポイントをまとめま

す。

- **モデル選定**: 音声とキャプションの整合度評価にはCLAPを用いる（スペクトログラム入力＋テキストエンコーダ）huggingface.co。音響詳細分析にはMERTを併用し、音声埋め込みを取得する。両モデルともPyTorch実装があり、HuggingFace経由で入手・利用可能github.comhuggingface.co。
- **前処理**: CLAPは音声を10秒窓・48kHzに揃えログメル化、MERTは30秒まで波形直接入力（24kHz）で処理arxiv.org。composer4データではループ長が短い想定のため、それぞれ標準の前処理設定で問題ない。日本語キャプションは事前に英訳しCLAPテキストエンコーダに投入する。
- **評価指標**: CLAPの音声・テキスト埋め込みのコサイン類似度（CLAPスコア）を **metrics.text_audio_cos** として算出arxiv.org。高スコアはキャプション妥当性を示す。MERTの音声埋め込みは **metrics.audio_embed_mert** として保存し、後続分析に活用。CLAPの音声埋め込みも **metrics.audio_embed_clap** で保持可能。
- **精度と効率**: CLAPはテキスト指向の高レベル特徴抽出に優れ、MERTは音響内容把握に優れるため相互補完的arxiv.orgarxiv.org。推論はキャッシュ戦略で効率化し、大量データでも実用的な処理時間を維持する。モデルロードは1回に集約し、埋め込み計算結果をIDやキャプション毎に再利用する。
- **実装統合**: Stage2抽出フェーズにオプション機能として組み込み、loop_summaryに新指標を追加する。既存CSVスキーマに影響しない範囲で列を増やし、Parquetで詳細ベクトルを保持する設計GitHub。CLI引数で音声入力先とキャプションファイルを指定できるようにし、未指定時は従来通りMIDI処理のみにもできる柔軟性を持たせる。
- **他モデルへの言及**: 必要に応じてT-CLAP等の新手法もウォッチし、時間的整合性評価が将来必要になれば導入を検討するarxiv.org。現段階ではCLAP+MERTで要件を満たすが、技術進展に応じてアップデート可能なアーキテクチャとして設計しておく。

以上の指針に従い、composer4のラベリングシステムに音声-テキスト整合モデルを組み込むことで、**音データと言語データを統合的に扱う高度なメタデータ付与**が実現できます。これにより、従来のMIDI解析軸に**「音そのものが語る情報」**という新たな軸が加わり、システム全体のラベル品質向上と検索性向上が期待できます。CLAPとMERTの強みを活かしつつ、将来的なモデル拡張にも対応できる柔軟な実装を目指してください。

参考文献・ソース: 本報告中で言及した内容の一部は、CLAPおよびMERTの公式資料や関連研究から引用しています。arxiv.orghuggingface.coarxiv.orgその他、モデルカードや論文から得た数値・事実を元に議論しました。各参照箇所の詳細については該当出典をご確認ください。