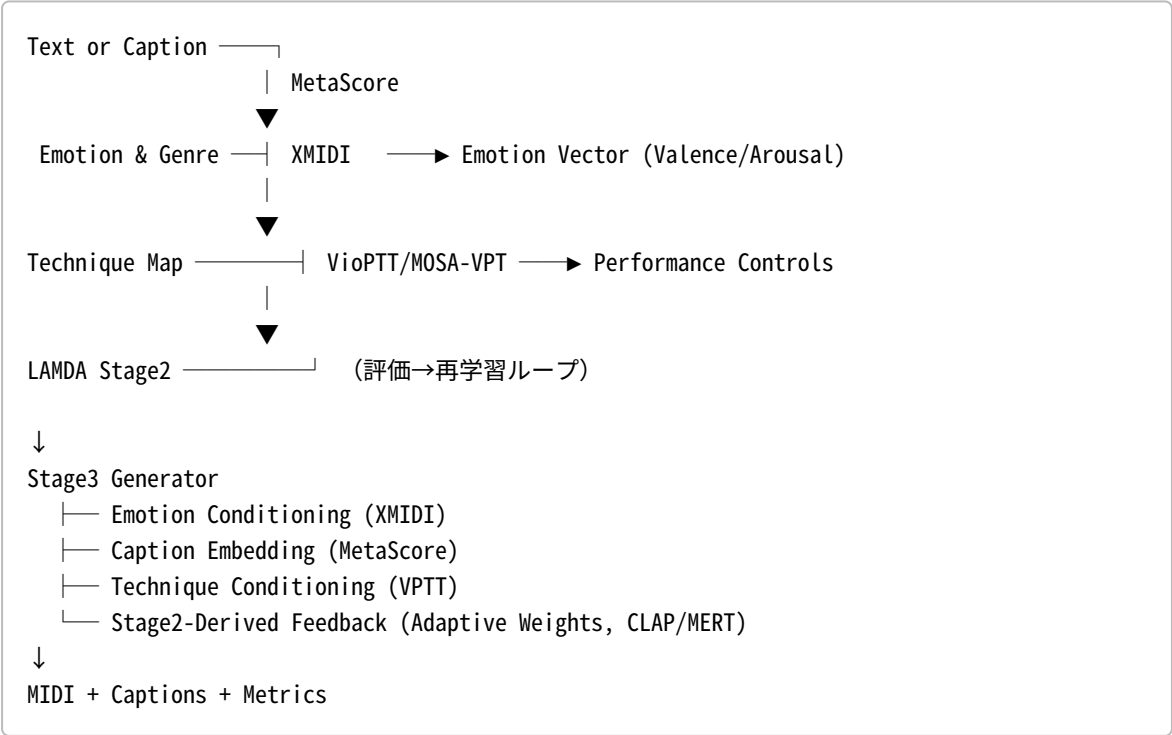


# Composer4 Stage3 ロードマップ (XMIDI・VioPTT/MOSA-VPT・MetaScore 統合)

## 🎯フェーズ概要

項目	説明
フェーズ名	Stage3: Context-Aware Composition Pipeline
目的	Stage2で得た「評価知能」を活用し、音楽生成（MIDI出力）へフィードバックする
技術中核	XMIDI（感情・ジャンル教師） + VioPTT/MOSA-VPT（奏法モデル） + MetaScore（テキストキャプション生成）
出力形式	Multi-Track MIDI（Emotion/Genre/Technique/Caption タグ付き） + Loop Summary + Auto Score
成功KPI	再現性 ↑（同属性で近似曲生成）／創造性維持／スコア p50 ≥ 75, p90 ≥ 88

## 🔗構造図（論理パイプライン）



## 実装ステップ

	フェーズ	実施内容	成果物
Step 1 : XMIDI 感情・ジャンル 教師導入		<code>labels_schema.yaml</code> を XMIDI 語彙に統一。 <code>assign_labels.py</code> で valence/arousal, genre を自動推定。	<code>loop_summary.csv</code> に <code>label.emotion</code> , <code>label.genre</code> 列を追加。
Step 2 : MetaScore キャ プション生成		<code>scripts/generate_music_captions.py</code> を作成。 Stage2 のメトリクス+歌詞/楽曲情報を LLM に入力。	<code>label.caption</code> 列を生成し、Emotion/Genre と連動。
Step 3 : VioPTT/ MOSA-VPT 奏法 制御導入		<code>configs/labels/technique_map.yaml</code> で奏法ラベルを定義。 <code>batch_articulation_renderer.py</code> で奏法合成。	合成データセット（奏法別 MIDI）+ 奏法マップ（Key/CC対応表）。
Step 4 : Stage3 Generator 開発		<code>ml/stage3_generator.py</code> : XMIDI・MetaScore・VPTT を条件トークンとして学習。	感情・テキスト・奏法条件付きの自動作曲モデル。
Step 5 : LAMDA Feedback 統合		Stage2 のメトリクスを Stage3 学習に報酬として取り込み（RL or weighting）。	自動評価・改善ループ（自己学習的生成）。

## 出力・レポート

出力物	説明
<code>stage3_summary.json</code>	Stage3生成の統計（Emotion, Caption, Technique別性能）。
<code>loop_summary_stage3.csv</code>	生成結果のメタ情報（感情・ジャンル・奏法・スコア）。
<code>music_captions.log</code>	MetaScore生成ログ（要約・情景文）。
<code>emotion_distribution.json</code>	感情ベクトルの分布ヒートマップ。
<code>stage3_ab_report.md</code>	A/B比較レポート（生成品質・評価スコア）。

## 成功指標（KPI）


軸	指標	目標値
品質	Stage2再評価後のp50/p90	≥ 75 / ≥ 88
文脈一致	CLAP/MERT類似度 (text_audio_cos)	≥ 0.65
感情再現率	Emotion分類精度 (XMIDI)	≥ 85%
奏法再現率	Techniqueラベル一致率	≥ 80%
創造的多様性	曲間平均類似度	≤ 0.55
再現性	同条件再生成のスコア分散	≤ 5%

---

## 今後の展望

- **MetaScore 2.0**：音楽構造を「物語の章」として説明できるキャプションを自動生成。
- **VPTT 拡張**：他楽器（Strings, Piano, Winds）への奏法転移学習。
- **XMIDI Fine-Tune**：自作データで valence/arousal の再学習、より日本語感情語彙へローカライズ。
- **Stage4 Preview**：自動リライト（「感情→旋律→調整→再評価」の完全生成ループ）。

---

 **結論**：Composer4 は Stage2 の“評価知能”を完成させ、次に Stage3 で“文脈と感情を理解する生成AI”へ進化する段階に入りました。

---

# 拡張パック連携仕様書（Extended Integration Spec v0.1）

Stage3 までを見据え、各拡張パックの **入出力（I/O）**・**設定（YAML）**・**評価指標（KPI）**・**CI ガード** をひと目で参照できる“実装用”仕様です。まずは最小可動を優先し、順次詳細化します。

## 0. 共通ポリシー

- すべての外部アーティファクトは `configs/` と `artifacts/` に集約（読み取りは必須・書き出しは任意）。
- 生成物は `outputs/<date>/` 配下に固定。大容量は Parquet 優先、CSV は人間閲覧用。
- 参照キーは `loop_id` / `hash_id` を原則とし、欠損時は処理を止めず **degrade gracefully**。
- CI 必須：スキーマ検証（YAML/JSON Schema）＋参照ファイル存在確認＋しきい値の保守範囲チェック。

---

## 1. CLAP / MERT（Audio-Text/Audio-Audio 一致）

### 目的

- Stage2: 評価・リトライの文脈信号（`audio.*`）として使用。
- Stage3: 生成器の条件ベクトル／報酬フィードバックとして使用。

### I/O

- **入力**：
- `artifacts/audio/alignment.jsonl` ... `{ loop_id, clap: { emb: [...], text_audio_cos, conf }, mert: { emb: [...], audio_audio_cos, conf } }`
- `artifacts/audio/captions.json` ... `{ loop_id: { caption_ja, caption_en, conf } }`
- **出力**：
- Stage2: `metrics_score.jsonl` に `audio.*` を付加、`audio_embeddings.parquet` を保存
- Stage3: `conditions/audio_vec.parquet` に集約（列：`loop_id, clap_512, mert_768, text_audio_cos`）

## 設定（最小）

```
# configs/lamda/audio/adaptive_weights.yaml
enabled: true
fusion:
  use_clap: true
  use_mert: true
  temperature: 0.6 # 過剰反応抑制
  pivot_ema_alpha: 0.25
caps:
  normalize:
    target_sum: 8.2
  max_total_delta: 0.40
  per_axis_max_delta: { velocity: 0.18, structure: 0.12 }
  missing_policy: last # noop|zero|last
cooldown:
  default: 2
  rules: { fast_strict_downbeat: 3 }
```

## KPI

- A/B で **pass\_rate** ↑ / **p50** ↑ に寄与（`ab_summarize_v2.py`）。
- **Failsafe breakdown**：Missing/LowConf/Cooldown の割合 ≤ 15%。

## CI ガード

- `validate_audio_adaptive_config.py`：enabled 型、min\_confidence 範囲、caps 設定、cooldown の整合性。

## 2. XMIDI + EMOPIA（Emotion / Genre 教師）

### 目的

- Emotion/Genre を **世界標準の語彙** へ正規化（XMIDI）。
- Valence/Arousal（EMOPIA）へ写像し、Humanizer とリトライの“期待分布”を切替。

### I/O

- **入力**：
- `artifacts/xmidi/classifier.pt`（推論器）
- `artifacts/emopia/va_map.json`（ラベル→V/A テーブル）
- **出力**：
- Stage2: `loop_summary.csv` に `label.genre, label.emotion, emotion.valence, emotion.arousal`
- Stage3: `conditions/emotion.parquet`（`loop_id, valence, arousal`）

### 設定スケッチ

```
# configs/labels/labels_schema.yaml
emotion_vocab: [happy, sad, angry, relaxed, tense]
```

```
genre_vocab: [rock, jazz, edm, funk, hiphop]
# configs/emotion_profile.yaml
va_to_humanizer:
  low_valence:
    velocity_curve: gentle
    swing: 0.57
  high_arousal:
    velocity_curve: aggressive
    microtiming_std_ms: 12
```

## KPI

- Emotion/Genre F1: **+5pp 以上** (XMIDI ベースライン比)。
- V/A に応じた **Velocity 分布の KL 距離↓** (期待プロファイルとの整合)。

## CI ガード

- 語彙の未知ラベル検知、V/A 範囲 [-1, +1] 検証。

---

## 3. VioPTT / MOSA-VPT (奏法合成)

### 目的

- 既存 MIDI から **奏法ラベル付き** データを合成し、Stage3 学習の条件 ([technique:\*]) を増強。

### I/O

- 入力: `configs/labels/technique_map.yaml` (音源ごとの keyswitch/CC 定義)
- 出力: `outputs/technique_synth/` に合成 MIDI & ラベル JSON ( `loop_id, technique[]` )

### ユーティリティ雛形

```
python scripts/daw/batch_articulation_renderer.py
--in loops/ --map configs/labels/technique_map.yaml
--out outputs/technique_synth
```

## KPI

- `technique` 条件での再現性 (同条件→同傾向) と、Stage2 Articulation の平均↑。

## CI ガード

- `technique_map` のキー/ノート競合、CC 範囲 (0-127) 検証。

## 4. MT3 + ASAP/nASAP（転写・演奏化）

### 目的

- 音源→MIDI 転写（MT3）と、スコア-演奏アライン（ASAP/nASAP）で **Humanizer/Rubato** を教師化。

### I/O

- 入力： `stems/*.wav`、リファレンス MIDI（任意）
- 出力： `outputs/transcription/*.mid`、テンポ曲線 `rubato.jsonl`

### KPI

- 転写 F1（note onset within 20ms）↑、過剰ゆらぎの抑制（テンポ曲線の TV 正則化↓）。

### CI ガード

- サンプリングレート/チャンネル整合、ファイルサイズ閾値。
- 

## 5. MetaScore + LP-MusicCaps（キャプション強化）

### 目的

- Stage2 メトリクス×歌詞×近傍キャプションから **短い日本語キャプション** を生成し、Stage3 条件へ。

### I/O

- 入力： `loop_summary.csv` の数値指標、`lyrics/*.txt`、`lp_musiccaps.index`
- 出力： `captions/*.jsonl` (`loop_id`, `caption_ja`, `attrs[]`)

### 設定スケッチ

```
# configs/caption/prompt.yaml
style: concise_ja
hints: [tempo, groove, intensity, texture]
```

### KPI

- Caption 一貫性（属性トークン化後の分布安定）／ABX 主観評価で選好率 > 50%。

### CI ガード

- 出力 JSONL スキーマ、空文字列の除去。
-

## 6. MuseCoco（属性トークン化）

### 目的

- 自由文を `[genre:jazz][mood:sad][tempo:fast]` の **属性列** に正規化し、再現性を上げる。

### I/O

- 入力: `captions/*.jsonl`
- 出力: `conditions/attrs.parquet` (`loop_id`, `tokens[]`)

### KPI

- 同一属性トークンでの再生成安定性（分散↓）。

### CI ガード

- 未知属性の拒否、値域（e.g., `tempo ∈ {slow, mid, fast}`）。
- 

## 7. A/B & 運用（横断）

- `ab_summarize_v2.py` を **PR アーティファクト** として生成：Overall KPI、Audio Adaptive、Retry Decision、Stratified。
  - `guard_retry_accept.py` で **score/axes + audio.cos\_delta** の複合判定を運用。
  - Dash 化候補**：velocity\_coverage.json（BPM 帯×ピンのカバレッジ率）を週次でプロット。
- 

## 8. 直近の導入順（推奨）

- CLAP/MERT（Stage2 実装済 → Stage3 条件化）
- XMIDI+EMOPIA（Emotion 正規化 & Humanizer 連携）
- MetaScore + MuseCoco（Caption→属性トークン化）
- VioPTT/MOSA-VPT（奏法データ拡充）
- MT3+ASAP（転写/Rubato 教師）

上記順番なら、既存 Stage2 の改善ループと親和性が高く、**pass\_rate** と生成再現性を同時に底上げできます。