CQOx_gen 拡張案 v2

Thought for 23s >

3行サマリ

1. v1 は「準備 → 結果」の平均効果はきれいに見えているけれど、「なぜ涙が出るのか」の因果ストーリーが UI 上まだ弱い。
2. v2 では **(A) 評価コンテキストと抑え込み意図のログ追加**、**(B) Trait プロファイル**、**(C) 評価→不安→抑え込み→涙 のパス推定＋可視化 UI** を入れて、「原因の地図」が見える形にする。
3. ①②はフル SEM/CLPM の上限として意識しつつ、v2 は「単純な回帰＋パス図＋ナラティブ生成」で、実装可能な範囲の“世界レベル”を取りに行く。

---

0. いまの v1 をどう見ているか（前提）

良いところ

- Episode Draft / Outcome Log / Dashboard の3画面はすでに「1→N→∞」の流れができている。
- Dashboard のフォレスト＋時系列は、「準備ごとの平均効果」と「波」をかなりきれいに見せている。
- コードベースも CQOx_gen と揃っていて、拡張しやすい構造。

足りていないところ（v2で狙う）

1. 「なぜ涙が出るのか」の**メカニズム視点**が UI にまだ出ていない。
   - 今は「準備の効果」だけで、「評価されそう」「抑え込もうとした」がどれだけ効いたかが見えない。
2. 「原因の説明」が**数字だけで、言葉になっていない**。
   - ATE の棒グラフは良いが、「あなたの場合、評価が厳しいと感じるほど、こういう経路で涙が出やすい」といった文章になっていない。
3. Trait（元からの泣きやすさ・社会不安）と Episode（その日の状態）が分かれておらず、
   - 「もともとの自分」と「今回の状況」が混ざってしまっている。

→ v2 は **「評価 → 不安 → 抑え込み → 涙」のパスをログ＆推定し、UIで“物語”として返す**ことを目標にします。

---

v2 のゴールと範囲

ゴール

1. Episode 層に
   - 評価コンテキスト（どれくらい厳しく評価されそうか）
   - 泣きを抑えようとする意図（どれくらい「泣いてはいけない」と思っているか）
   を追加し、
   「評価の恐さ → 不安 → 抑え込み → 涙レベル」のパスを推定できるようにする。
2. ユーザーごとに
   - そのパスの強さ（係数）と
   - そこに準備がどう割り込むか
   を、UI で **図＋文章**として見せる。
3. Trait（元々の社会不安・泣きやすさ・抑制傾向）を別に持ち、
   Episode の回帰では confounder として調整する。

やらないこと（v2 の Non-goals）

- 完全な SEM / CLPM 実装（潜在変数・縦断の厳密検証）はやらない。
  → ①②の「理想像」に対して、v2 は **観測変数でのパス回帰＋ATE** に留める。
- 医療・診断レベルのラベル・フィードバックは出さない。
  → UI 文言は「傾向」「パターン」「推定」止まりにする。

Change Set A: Episode に評価コンテキスト＋抑え込み意図を追加

A-1. 何を足すか

Episode レコードに 2 つのスライダー値

- `eval_threat_level`：
  「どれくらい厳しく評価されそうだと感じますか？」（0–10）

- `suppress_intent_level`：
  「どれくらい『泣かないようにしよう』と思っていますか？」（0–10）

これで DAG の E と R を、ひとまず観測変数として持てる。

A-2. DB スキーマ差分

```
ALTER TABLE emotion_episode
  ADD COLUMN eval_threat_level INTEGER,
  ADD COLUMN suppress_intent_level INTEGER;
```

A-3. SQLAlchemy モデル差分

```
# backend/cqox/emotion/models.py
class EmotionEpisode(Base):
    __tablename__ = "emotion_episode"
    # ...既存...

    eval_threat_level = Column(Integer, nullable=True)       # v2 追加
    suppress_intent_level = Column(Integer, nullable=True)   # v2 追加
```

A-4. Pydantic schema 差分

```
# backend/cqox/emotion/schemas.py
from pydantic import conint

class EpisodeDraftCreate(BaseModel):
    scenario_type: str
    topic: str
    scheduled_at: datetime
    location: str
    pre_state: PreState
    preparations_planned: PreparationPlan
    preference_weights_raw: PreferenceWeightsRaw

    # v2 追加
    eval_threat_level: conint(ge=0, le=10)
    suppress_intent_level: conint(ge=0, le=10)
```

A-5. Episode Draft UI への追加（重要：UIの質）

Layer C の右カラム、「準備プランのシミュレーション」の下に「評価される怖さ」「泣きを抑えたい度」を入れると流れが自然です。

```
// EmotionEpisodeCreatePage.tsx（一部）
const [evalThreat, setEvalThreat] = useState(5);
const [suppressIntent, setSuppressIntent] = useState(5);

// payload に含める
const payload = {
  // ...既存...
  eval_threat_level: evalThreat,
  suppress_intent_level: suppressIntent,
};
```

UI コンポーネント案：

```
// frontend/src/features/emotion/components/EvaluationContextSliders.tsx
export const EvaluationContextSliders: React.FC<{
  evalThreat: number;
  suppressIntent: number;
  onEvalThreatChange: (v: number) => void;
  onSuppressIntentChange: (v: number) => void;
}> = ({ evalThreat, suppressIntent, onEvalThreatChange, onSuppressIntentChange }) => (
  <div className="mt-6 space-y-4">
    <div>
      <p className="text-sm font-semibold">どれくらい厳しく評価されそうだと感じますか？</p>
```

```
      <p className="text-xs text-gray-500 mb-2">
        0：ほとんど評価されない / 10：とても厳しくジャッジされそう
      </p>
      <Slider value={evalThreat} min={0} max={10} onChange={onEvalThreatChange} />
    </div>
    <div>
      <p className="text-sm font-semibold">どれくらい「泣いてはいけない」と思っていますか？</p>
      <p className="text-xs text-gray-500 mb-2">
        0：特に意識していない / 10：絶対に泣いてはいけないと強く思う
      </p>
      <Slider value={suppressIntent} min={0} max={10} onChange={onSuppressIntentChange} />
    </div>
  </div>
);
```

呼び出し：

```
<EvaluationContextSliders
  evalThreat={evalThreat}
  suppressIntent={suppressIntent}
  onEvalThreatChange={setEvalThreat}
  onSuppressIntentChange={setSuppressIntent}
/>
```

A-6. コマンド

```
# マイグレーション反映
alembic revision -m "add eval_threat_and_suppress_intent"
alembic upgrade head
```

Change Set B: Trait プロファイルを持たせる

B-1. 目的

- 「もともと泣きやすい」「もともと社会不安が高い」を Episode とは別に持ちたい。
- これを confounder として回帰に入れることで、「その日の評価の恐さ」の効果を少しでもクリーンにする。

B-2. DB

```
CREATE TABLE emotion_trait_profile (
  user_id                INTEGER PRIMARY KEY,
  trait_social_anxiety   INTEGER NOT NULL,
  trait_crying_proneness INTEGER NOT NULL,
  trait_suppression      INTEGER NOT NULL,
  updated_at             TIMESTAMP NOT NULL DEFAULT now()
);
```

モデルは前回案と同じなので省略。

B-3. UI：とても軽い設定画面にする

ここは「診断っぽさ」を出さず、**自己感覚をざっくり入れるだけ**にします。

- `/emotion/settings` 画面：
  - 「人前で話すときの不安の強さ（普段の平均）」
  - 「もともとの泣きやすさ」
  - 「普段どれくらい感情を抑えがちか」
- 1回入力したら、たまに見直す程度。

Change Set C: 評価→不安→抑え込み→涙 のパス推定＋UI

ここが「原因が見える」部分。

C-1. モデルの考え方（私の判断）

①②だと SEM/CLPM まで視野に入っているけど、v2 でやり切るのは現実的ではないので、**以下の折衷案**にします：

- ひとりのユーザーについて、
  - `E = eval_threat_level`
  - `S = pre_anxiety`（今のしんどさ）
  - `R = suppress_intent_level`
  - `C = crying_level`
  - Trait: `A_sa, A_cp`
- 2本の線形回帰を使った**シンプルなパスモデル**：
  1. $S = \alpha_0 + \alpha_1 E + \alpha_2 A_{sa} + \epsilon_S$
  2. $C = \beta_0 + \beta_1 E + \beta_2 S + \beta_3 R + \beta_4 A_{cp} + \epsilon_C$
- 「評価の恐さが涙に与える**間接効果**」を $\alpha_1 \times \beta_2$ として出す。
- 「直接効果」は β_1。

理由（私の判断）

- Trait を明示的に入れることで、「もともと不安が高い人が評価も高く付けがち」というバイアスを少し抑えられる。
- 2本の回帰だけなので、データ40〜50 Episode でも数値はそこそこ安定する。
- UI で見せるときに「評価 → 不安 → 涙」の矢印をそのまま数値に割り当てられる。

C-2. 実装（ジョブ）

新テーブル：

```
CREATE TABLE emotion_path_summary (
  id                    SERIAL PRIMARY KEY,
  user_id               INTEGER NOT NULL,
  alpha_eval_to_stress  DOUBLE PRECISION,
  beta_eval_to_cry      DOUBLE PRECISION,
  beta_stress_to_cry    DOUBLE PRECISION,
  beta_suppress_to_cry  DOUBLE PRECISION,
  indirect_eval_to_cry  DOUBLE PRECISION,
  total_eval_to_cry     DOUBLE PRECISION,
  n_episodes            INTEGER NOT NULL,
  updated_at            TIMESTAMP NOT NULL DEFAULT now(),
  UNIQUE (user_id)
);
```

ジョブ（ざっくり版）：

```
# backend/cqox/jobs/estimate_paths.py
import pandas as pd
from sqlalchemy.orm import Session
from sklearn.linear_model import LinearRegression
from datetime import datetime

from cqox.db import SessionLocal
from cqox.emotion import models


def build_user_df(db: Session) -> pd.DataFrame:
    q = (
        db.query(models.EmotionEpisode, models.EmotionOutcome, models.EmotionTraitProfile)
        .join(models.EmotionOutcome, models.EmotionEpisode.id == models.EmotionOutcome.episode_id)
        .outerjoin(models.EmotionTraitProfile,
                   models.EmotionTraitProfile.user_id == models.EmotionEpisode.user_id)
    )
    rows = []
    for ep, oc, tr in q:
        if ep.eval_threat_level is None or ep.suppress_intent_level is None:
            continue
        rows.append({
            "user_id": ep.user_id,
            "E": ep.eval_threat_level,
            "S": ep.pre_anxiety,
            "R": ep.suppress_intent_level,
            "C": oc.crying_level,
```

```
            "A_sa": tr.trait_social_anxiety if tr else None,
            "A_cp": tr.trait_crying_proneness if tr else None,
        })
    return pd.DataFrame(rows)


def estimate_for_user(df_u: pd.DataFrame):
    df_u = df_u.dropna()
    if len(df_u) < 15:
        return None

    # 1) S ~ E + A_sa
    X_s = df_u[["E", "A_sa"]].fillna(df_u[["E", "A_sa"]].mean())
    y_s = df_u["S"].values
    lr_s = LinearRegression().fit(X_s, y_s)
    alpha_eval = float(lr_s.coef_[0])

    # 2) C ~ E + S + R + A_cp
    X_c = df_u[["E", "S", "R", "A_cp"]].fillna(df_u[["E", "S", "R", "A_cp"]].mean())
    y_c = df_u["C"].values
    lr_c = LinearRegression().fit(X_c, y_c)
    beta_eval = float(lr_c.coef_[0])
    beta_stress = float(lr_c.coef_[1])
    beta_suppress = float(lr_c.coef_[2])

    indirect = alpha_eval * beta_stress
    total = beta_eval + indirect

    return {
        "alpha_eval_to_stress": alpha_eval,
        "beta_eval_to_cry": beta_eval,
        "beta_stress_to_cry": beta_stress,
        "beta_suppress_to_cry": beta_suppress,
        "indirect_eval_to_cry": indirect,
        "total_eval_to_cry": total,
        "n_episodes": len(df_u),
    }


def run():
    db = SessionLocal()
    try:
        df = build_user_df(db)
        if df.empty:
            return
        for user_id, df_u in df.groupby("user_id"):
            stats = estimate_for_user(df_u)
            if stats is None:
                continue
            obj = (
                db.query(models.EmotionPathSummary)
                .filter_by(user_id=user_id)
                .one_or_none()
            )
            if obj is None:
                obj = models.EmotionPathSummary(user_id=user_id)
                db.add(obj)

            for k, v in stats.items():
                setattr(obj, k, v)
            obj.updated_at = datetime.utcnow()
        db.commit()
    finally:
        db.close()


if __name__ == "__main__":
    run()
```

C-3. UI：Path Insight カード（ここが一番 UI 的に大事）

Dashboard のフォレストの上に、「あなたのパターン（評価と涙の関係）」のカードを追加します。

```tsx
// frontend/src/features/emotion/components/EvalToCryingPathCard.tsx
type PathSummary = {
  alpha_eval_to_stress: number;
  beta_eval_to_cry: number;
  beta_stress_to_cry: number;
  beta_suppress_to_cry: number;
  indirect_eval_to_cry: number;
  total_eval_to_cry: number;
  n_episodes: number;
};

export const EvalToCryingPathCard: React.FC<{ data: PathSummary }> = ({ data }) => {
  const { indirect_eval_to_cry: ind, total_eval_to_cry: total } = data;

  const evalText =
    total > 0.3
      ? "評価されるほど、涙が増えやすい傾向があります。"
      : total < -0.3
      ? "評価が厳しいほど、むしろ涙が減る珍しいパターンです。"
      : "評価の厳しさと涙レベルの関係は、今のところ強くはありません。";

  const pathText =
    Math.abs(ind) > 0.3
      ? "特に「評価 → 今のしんどさ → 涙」という経路の影響が大きいです。"
      : "今回のデータでは、「評価 → しんどさ → 涙」の経路はそこまで強くありません。";

  const suppressText =
    data.beta_suppress_to_cry > 0.3
      ? "「泣かないようにしよう」と強く思うほど、かえって涙レベルが上がる傾向があります。"
      : data.beta_suppress_to_cry < -0.3
      ? "「泣かないようにしよう」という意図は、ある程度涙を下げる方向に働いています。"
      : "抑え込みの意図と涙レベルの関係は、今のところはっきりしていません。";

  return (
    <div className="border rounded-2xl p-4 shadow-sm bg-white">
      <h2 className="text-sm font-semibold mb-2">評価の恐さと涙の関係（あなたのパターン）</h2>
      <p className="text-xs text-gray-600 mb-1">
        エピソード {data.n_episodes} 件のデータから、「評価の厳しさ → 今のしんどさ → 涙レベル」の関係を推定しています。
      </p>
      <ul className="text-sm space-y-1 mt-2">
        <li>・評価→涙の合計効果（1ポイント厳しく感じた時の平均変化）：<b>{total.toFixed(2)}</b></li>
        <li>・評価→しんどさ→涙の間接効果：<b>{ind.toFixed(2)}</b></li>
        <li>・「泣かないようにしよう」という意図→涙の変化：<b>{data.beta_suppress_to_cry.toFixed(2)}</b></li>
      </ul>
      <div className="mt-3 text-sm space-y-1">
        <p>{evalText}</p>
        <p>{pathText}</p>
        <p>{suppressText}</p>
      </div>
      <p className="mt-2 text-[10px] text-gray-400">
        ※ あくまであなたのログに基づく統計的な推定です。診断や保証ではありません。
      </p>
    </div>
  );
};
```

Dashboard レイアウト：

- 上段：
  - 左：`EvalToCryingPathCard`
  - 右：既存の Episode 数 / よく実施している準備タグ
- 中段：既存フォレスト（涙レベル / 伝えられた感）
- 下段：時系列グラフ

C-4. これで「原因はどこにあるのか？」への答え方が変わる

UI 的には、ユーザー視点でこう見える：

- 「評価の恐さが 1 段階上がると、平均で涙レベルが +0.8 上がる（うち +0.6 は“しんどさ経由”）」
- 「抑え込もうとすると +0.5 上がっているので、自分には逆効果かもしれない」

→ v1 の「準備ごとの効果」だけではなく、
「評価状況そのものがどれくらい涙を引き起こしているか」と
「抑え込み戦略が自分にとって毒なのか薬なのか」まで見える。

What this does（今回の v2 提案で実現すること）

- **Goal**
  - Emotion CQOx を「準備の効果を見るツール」から、
    「評価の恐さ → 不安 → 抑え込み → 涙」の因果パターンまで見えるツールへアップグレードする。
- **Inputs**
  - Episode Draft で入力する：
    - 今のしんどさ（既存）
    - 評価の厳しさ（新）
    - 「泣かないようにしよう」という意図（新）
  - Outcome Log で入力する：
    - 涙レベル（既存）
  - Settings で入力する：
    - 元々の社会不安 / 泣きやすさ / 抑制傾向（新）
- **Outputs / side-effects**
  - 新ジョブ `estimate_paths.py` により、ユーザーごとに
    - 評価→不安、評価→涙、しんどさ→涙、抑え込み→涙 の係数
    - 評価→涙の間接効果・合計効果
      が `emotion_path_summary` に保存される。
  - Dashboard の `EvalToCryingPathCard` で、
    - 数値＋ミニ解釈文で「あなたのパターン」が返る。
  - 既存のフォレスト・時系列は維持され、
    - 「準備」の因果効果と「評価コンテキスト」の因果パスが両方見える状態になる。
- **How to run / verify**
  1. マイグレーションを当てる（Change Set A/B/C のテーブル追加）。
  2. Episode Draft で新しい2つの slider を含めて 20〜30 Episode 入力する。
  3. Trait 設定を仮に入力する。
  4. `python -m cqox.jobs.estimate_paths` を実行し、`emotion_path_summary` に行ができるか確認。
  5. `/emotion/dashboard` を開き、
     - Path カードに数値と文章が表示されること
     - 棒グラフ・時系列との整合性（例：評価が高い場面ほど涙レベルが高い）が直感と合うか
       をチェックする。

Expert insight (Google/Meta/NASA level)

世界レベルのプロダクトは、「複雑な理論モデル全部を実装する」のではなく、「ユーザーの問いに直結するパスだけを、UI で説明できるレベルまで落とし込む」ことを優先します。

- ①②の SEM / CLPM の世界観は「どこまで行けるか」という上限値として頭に置きつつ、
  v2 では「評価→不安→抑え込み→涙」の1本に集中し、
  - ログ
  - モデル
  - UI テキスト
    を全部この線に揃えた。
- こういう**“1本のストーリーに全スタックを合わせる”**設計をすると、
  - コードも

- UI も
- ユーザーの体験も
  ずれずに成長していきやすくなります。