CQOx_gen 拡張案④

Thought for 46s >

先に前提だけはっきりさせます。

- 「世界最高峰・これ以上無理」は、実務的には常に更新され続けるので**絶対値として保証はできない**です。
- ただ、**2025 時点の因果推論 × 個人向け支援ツールとして、これ以上は「別プロダクト」になるだろう、というライン**までは攻めて設計します。

以下は

1. 本番アーキテクチャ
2. DB と因果パイプライン
3. API
4. フロント（可視化を含む）
5. 実行・検証方法

を、コード付きでまとめます。

---

0. 全体像（最終本番版）

プロダクト定義

- 目的：
  「**人前で自分の話をする時に、涙・しんどさ・伝えられなさがどう動くのかを、その人自身のデータから因果的に理解し、次の一回を少しマシにするプランを出す**」
- 単位：Episode（面接・1on1・大事な話し合いなど）
- ログ：
  - 前：状態＋準備プラン
  - 後：実際の感情と関係の変化
- 解析：
  - per-user の因果効果（ジャーナリング/3メッセージ/呼吸/ロールプレイ/セーフワード…）
  - 一部は意図的ランダム化で N-of-1 RCT を回す
- 出力：
  - 1 回ごとのフィードバックカード
  - 「あなたのパターン」因果ダッシュボード
  - 準備プランのレコメンド＋シミュレーション

---

1. アーキテクチャ

1.1 コンポーネント

- **フロントエンド**
  - React + Vite + TypeScript
  - TanStack Query（API データ取得）
  - Recharts（可視化）
- **バックエンド API**
  - FastAPI (Python)
  - 認証（JWT / OAuth2, ここでは省略）
- **ストレージ**
  - PostgreSQL（本番向け）
- **因果推論/バッチ**
  - Python（pandas, scikit-learn ベース）
  - `jobs/estimate_effects.py` として定期実行（Cron / Airflow / Prefect など）
- **メトリクス・監視**
  - Prometheus + Grafana（API レイテンシ・バッチの成功/失敗など）

ディレクトリ例：

```
emotion-cqox/
  backend/
    cqox/
      emotion/
        models.py
        schemas.py
        api.py
        service.py
      jobs/
        estimate_effects.py
    tests/
      test_emotion_api.py
  frontend/
    src/
      pages/
        EmotionEpisodeCreatePage.tsx
        EmotionOutcomeLogPage.tsx
        EmotionDashboardPage.tsx
      features/emotion/
        components/
          EpisodeQuickSliders.tsx
          PreferenceSliders.tsx
          PreparationSimulator.tsx
          CausalEffectForestChart.tsx
          EmotionTimelineChart.tsx
          EffectSummaryCards.tsx
```

---

2. データモデル（DB / models.py）

2.1 SQL スキーマ（PostgreSQL）

```sql
-- Episode 本体
CREATE TABLE emotion_episode (
  id                SERIAL PRIMARY KEY,
  user_id           INTEGER NOT NULL,
  scenario_type     VARCHAR(32) NOT NULL,
  topic             VARCHAR(128) NOT NULL,
  scheduled_at      TIMESTAMP NOT NULL,
  location          VARCHAR(32) NOT NULL,
  status            VARCHAR(16) NOT NULL, -- planned / completed / cancelled

  pre_anxiety               INTEGER NOT NULL,
  pre_crying_risk           INTEGER NOT NULL,
  pre_speech_block_risk     INTEGER NOT NULL,

  created_at        TIMESTAMP NOT NULL DEFAULT now(),
  updated_at        TIMESTAMP NOT NULL DEFAULT now()
);

-- 準備アクション（1 Episode に複数行）
CREATE TABLE emotion_preparation_execution (
  id               SERIAL PRIMARY KEY,
  episode_id       INTEGER NOT NULL REFERENCES emotion_episode(id) ON DELETE CASCADE,
  template_key     VARCHAR(64) NOT NULL, -- journaling_10m, three_messages, ...
  planned_intensity INTEGER NOT NULL,    -- 0-10
  actual_intensity  INTEGER,             -- 0-10, 終了後に自己評価
  created_at       TIMESTAMP NOT NULL DEFAULT now()
);

-- Episode のアウトカム
CREATE TABLE emotion_outcome (
  episode_id            INTEGER PRIMARY KEY REFERENCES emotion_episode(id) ON DELETE CASCADE,
  stress_during         INTEGER NOT NULL,
  stress_after          INTEGER NOT NULL,
  crying_level          INTEGER NOT NULL,
```

```
  speech_block_level    INTEGER NOT NULL,
  expression_score      INTEGER NOT NULL,
  relationship_impact   INTEGER NOT NULL, -- -5〜+5
  partner_reaction      VARCHAR(32),
  days_after_reflection INTEGER,
  would_repeat_preparation INTEGER,
  reflection_short      TEXT,
  created_at            TIMESTAMP NOT NULL DEFAULT now()
);

-- ユーザーの目的関数（重み）
CREATE TABLE emotion_preference_profile (
  user_id               INTEGER PRIMARY KEY,
  weight_relief         NUMERIC(4,3) NOT NULL,
  weight_expression     NUMERIC(4,3) NOT NULL,
  weight_relationship   NUMERIC(4,3) NOT NULL,
  updated_at            TIMESTAMP NOT NULL DEFAULT now()
);

-- 因果効果の推定結果（ユーザー毎 × 準備毎）
CREATE TABLE emotion_treatment_effect (
  id                    SERIAL PRIMARY KEY,
  user_id               INTEGER NOT NULL,
  treatment_key         VARCHAR(64) NOT NULL, -- journaling_10m, three_messages, ...
  outcome_name          VARCHAR(64) NOT NULL, -- crying_level, stress_after, ...
  ate                   DOUBLE PRECISION NOT NULL, -- 推定平均効果
  ci_lower              DOUBLE PRECISION,
  ci_upper              DOUBLE PRECISION,
  n_treated             INTEGER NOT NULL,
  n_control             INTEGER NOT NULL,
  model_version         VARCHAR(32) NOT NULL,
  updated_at            TIMESTAMP NOT NULL DEFAULT now(),
  UNIQUE (user_id, treatment_key, outcome_name)
);
```

2.2 SQLAlchemy モデル（抜粋）

```
# backend/cqox/emotion/models.py
from sqlalchemy import (
    Column, Integer, String, DateTime, ForeignKey, Numeric
)
from sqlalchemy.orm import relationship
from datetime import datetime

from cqox.db import Base  # Declarative base

class EmotionEpisode(Base):
    __tablename__ = "emotion_episode"

    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, nullable=False)
    scenario_type = Column(String(32), nullable=False)
    topic = Column(String(128), nullable=False)
    scheduled_at = Column(DateTime, nullable=False)
    location = Column(String(32), nullable=False)
    status = Column(String(16), nullable=False)

    pre_anxiety = Column(Integer, nullable=False)
    pre_crying_risk = Column(Integer, nullable=False)
    pre_speech_block_risk = Column(Integer, nullable=False)

    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)
    updated_at = Column(
        DateTime, default=datetime.utcnow, onupdate=datetime.utcnow, nullable=False
    )

    preparations = relationship("EmotionPreparationExecution", back_populates="episode")
    outcome = relationship("EmotionOutcome", uselist=False, back_populates="episode")
```

```python
class EmotionPreparationExecution(Base):
    __tablename__ = "emotion_preparation_execution"

    id = Column(Integer, primary_key=True)
    episode_id = Column(Integer, ForeignKey("emotion_episode.id", ondelete="CASCADE"))
    template_key = Column(String(64), nullable=False)
    planned_intensity = Column(Integer, nullable=False)
    actual_intensity = Column(Integer)

    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    episode = relationship("EmotionEpisode", back_populates="preparations")


class EmotionOutcome(Base):
    __tablename__ = "emotion_outcome"

    episode_id = Column(
        Integer, ForeignKey("emotion_episode.id", ondelete="CASCADE"),
        primary_key=True
    )
    stress_during = Column(Integer, nullable=False)
    stress_after = Column(Integer, nullable=False)
    crying_level = Column(Integer, nullable=False)
    speech_block_level = Column(Integer, nullable=False)
    expression_score = Column(Integer, nullable=False)
    relationship_impact = Column(Integer, nullable=False)
    partner_reaction = Column(String(32))
    days_after_reflection = Column(Integer)
    would_repeat_preparation = Column(Integer)
    reflection_short = Column(String)

    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    episode = relationship("EmotionEpisode", back_populates="outcome")


class EmotionPreferenceProfile(Base):
    __tablename__ = "emotion_preference_profile"

    user_id = Column(Integer, primary_key=True)
    weight_relief = Column(Numeric(4, 3), nullable=False)
    weight_expression = Column(Numeric(4, 3), nullable=False)
    weight_relationship = Column(Numeric(4, 3), nullable=False)
    updated_at = Column(DateTime, default=datetime.utcnow, nullable=False)


class EmotionTreatmentEffect(Base):
    __tablename__ = "emotion_treatment_effect"

    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, nullable=False)
    treatment_key = Column(String(64), nullable=False)
    outcome_name = Column(String(64), nullable=False)
    ate = Column(Numeric, nullable=False)
    ci_lower = Column(Numeric)
    ci_upper = Column(Numeric)
    n_treated = Column(Integer, nullable=False)
    n_control = Column(Integer, nullable=False)
    model_version = Column(String(32), nullable=False)
    updated_at = Column(DateTime, default=datetime.utcnow, nullable=False)
```

---

3. 因果パイプライン（jobs/estimate_effects.py）

3.1 目標

- 各ユーザー u, 各準備 `treatment_key` に対して

- Outcome（`crying_level`, `stress_after`, `expression_score`, `relationship_impact`）への**平均処置効果 ATE_u** とその CI を推定して `emotion_treatment_effect` に書き込む。
- モデルは最初はシンプルな DML 風（線形 + ランダムフォレスト）くらいから。

3.2 コア処理コード（簡略版）

```
# backend/cqox/jobs/estimate_effects.py
import math
from typing import List, Dict

import numpy as np
import pandas as pd
from sqlalchemy.orm import Session
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression

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


TREATMENTS = [
    "journaling_10m",
    "three_messages",
    "breathing_4_7_8",
    "roleplay_self_qa",
    "safe_word_plan",
]

OUTCOMES = [
    "crying_level",
    "stress_after",
    "expression_score",
    "relationship_impact",
]

MODEL_VERSION = "v1.0-dml-rf-lr"


def load_episode_dataframe(db: Session) -> pd.DataFrame:
    """DB から Episode + Preparation + Outcome をまとめたフラットな DF を作る。"""
    episodes = (
        db.query(models.EmotionEpisode)
        .join(models.EmotionOutcome)
        .all()
    )

    rows: List[Dict] = []
    for ep in episodes:
        if ep.status != "completed" or not ep.outcome:
            continue
        base = {
            "episode_id": ep.id,
            "user_id": ep.user_id,
            "scenario_type": ep.scenario_type,
            "topic": ep.topic,
            "scheduled_at": ep.scheduled_at,
            "location": ep.location,
            "pre_anxiety": ep.pre_anxiety,
            "pre_crying_risk": ep.pre_crying_risk,
            "pre_speech_block_risk": ep.pre_speech_block_risk,
            "crying_level": ep.outcome.crying_level,
            "stress_after": ep.outcome.stress_after,
            "expression_score": ep.outcome.expression_score,
            "relationship_impact": ep.outcome.relationship_impact,
        }
        # 準備強度
        intensity = {f"prep_{k}_intensity": 0 for k in TREATMENTS}
        for prep in ep.preparations:
            if prep.template_key in TREATMENTS:
```

```
                intensity[f"prep_{prep.template_key}_intensity"] = (
                    prep.actual_intensity
                    if prep.actual_intensity is not None
                    else prep.planned_intensity
                )
        rows.append({**base, **intensity})
    return pd.DataFrame(rows)


def dml_ate(
    df: pd.DataFrame,
    treatment_col: str,
    outcome_col: str,
    confounder_cols: List[str],
) -> Dict[str, float]:
    """
    ごく簡単な DML 風推定:
    - Y ~ X を RF で予測
    - T ~ X を線形で予測
    - 残差を取って、resY ~ resT の線形回帰から ATE を出す
    """

    d = df.copy()
    d = d.dropna(subset=[treatment_col, outcome_col] + confounder_cols)
    if len(d) < 30:
        return {"ate": 0.0, "se": math.inf, "n_treated": 0, "n_control": 0}

    T = d[treatment_col].values.astype(float)
    Y = d[outcome_col].values.astype(float)

    X = pd.get_dummies(
        d[confounder_cols],
        columns=["scenario_type", "location", "topic"],
        drop_first=True,
    )

    # 連続 treatment を 0/1 に単純二値化（強度 >= 3 を 1）
    T_bin = (T >= 3).astype(float)

    # Y ~ X
    y_model = RandomForestRegressor(
        n_estimators=200, max_depth=6, min_samples_leaf=10, n_jobs=-1
    )
    y_model.fit(X, Y)
    y_hat = y_model.predict(X)
    res_y = Y - y_hat

    # T ~ X
    t_model = LinearRegression()
    t_model.fit(X, T_bin)
    t_hat = t_model.predict(X)
    res_t = T_bin - t_hat

    lr = LinearRegression()
    lr.fit(res_t.reshape(-1, 1), res_y)
    ate = float(lr.coef_[0])

    # 標準誤差（ホワイト標準誤差っぽく）
    y_pred = lr.predict(res_t.reshape(-1, 1))
    resid = res_y - y_pred
    sigma2 = np.mean(resid ** 2)
    se = float(math.sqrt(sigma2 / np.sum((res_t - res_t.mean()) ** 2)))

    n_treated = int(T_bin.sum())
    n_control = int((1 - T_bin).sum())

    return {
        "ate": ate,
        "se": se,
```

```
        "n_treated": n_treated,
        "n_control": n_control,
    }


def estimate_and_persist_effects():
    db = SessionLocal()
    try:
        df = load_episode_dataframe(db)
        if df.empty:
            return

        for user_id, df_u in df.groupby("user_id"):
            for t_key in TREATMENTS:
                t_col = f"prep_{t_key}_intensity"
                for outcome in OUTCOMES:
                    stats = dml_ate(
                        df_u,
                        treatment_col=t_col,
                        outcome_col=outcome,
                        confounder_cols=[
                            "pre_anxiety",
                            "pre_crying_risk",
                            "pre_speech_block_risk",
                            "scenario_type",
                            "location",
                            "topic",
                        ],
                    )
                    if math.isinf(stats["se"]) or stats["n_treated"] == 0 or stats["n_control"] == 0:
                        continue

                    z = 1.96
                    ci_lower = stats["ate"] - z * stats["se"]
                    ci_upper = stats["ate"] + z * stats["se"]

                    eff = (
                        db.query(models.EmotionTreatmentEffect)
                        .filter_by(
                            user_id=user_id,
                            treatment_key=t_key,
                            outcome_name=outcome,
                        )
                        .one_or_none()
                    )
                    if eff is None:
                        eff = models.EmotionTreatmentEffect(
                            user_id=user_id,
                            treatment_key=t_key,
                            outcome_name=outcome,
                        )
                        db.add(eff)

                    eff.ate = stats["ate"]
                    eff.ci_lower = ci_lower
                    eff.ci_upper = ci_upper
                    eff.n_treated = stats["n_treated"]
                    eff.n_control = stats["n_control"]
                    eff.model_version = MODEL_VERSION

        db.commit()
    finally:
        db.close()


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

実行コマンド例

```
# バッチを単発実行
cd backend
python -m cqox.jobs.estimate_effects

# pytest で最低限の検証
pytest tests/test_emotion_api.py -q
```

---

4. API（FastAPI）

4.1 Episode 作成（ドラフト保存）

```
# backend/cqox/emotion/schemas.py
from datetime import datetime
from typing import Dict, Optional
from pydantic import BaseModel, conint, root_validator


class PreState(BaseModel):
    pre_anxiety: conint(ge=0, le=10)
    pre_crying_risk: conint(ge=0, le=10)
    pre_speech_block_risk: conint(ge=0, le=10)


class PreparationPlan(BaseModel):
    journaling_10m: conint(ge=0, le=10)
    three_messages: conint(ge=0, le=10)
    breathing_4_7_8: conint(ge=0, le=10)
    roleplay_self_qa: conint(ge=0, le=10)
    safe_word_plan: conint(ge=0, le=10)


class PreferenceWeightsRaw(BaseModel):
    relief: conint(ge=0, le=10)
    expression: conint(ge=0, le=10)
    relationship: conint(ge=0, le=10)

    @root_validator
    def ensure_nonzero(cls, values):
        if values["relief"] + values["expression"] + values["relationship"] == 0:
            # デフォルトは等分
            values["relief"] = values["expression"] = values["relationship"] = 1
        return values

    def normalized(self) -> Dict[str, float]:
        s = float(self.relief + self.expression + self.relationship)
        return {
            "relief": self.relief / s,
            "expression": self.expression / s,
            "relationship": self.relationship / s,
        }


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


class EpisodeDraftRead(BaseModel):
```

```
    episode_id: int
    normalized_weights: Dict[str, float]
```

```
# backend/cqox/emotion/api.py
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session

from cqox.dependencies import get_db, get_current_user
from . import schemas, service

router = APIRouter(prefix="/api/emotion", tags=["emotion"])


@router.post("/episodes/draft", response_model=schemas.EpisodeDraftRead)
def create_episode_draft(
    draft: schemas.EpisodeDraftCreate,
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    return service.create_episode_draft(db, user.id, draft)
```

4.2 Outcome 登録

```
# backend/cqox/emotion/schemas.py (続き)
class OutcomeCreate(BaseModel):
    stress_during: conint(ge=0, le=10)
    stress_after: conint(ge=0, le=10)
    crying_level: conint(ge=0, le=10)
    speech_block_level: conint(ge=0, le=10)
    expression_score: conint(ge=0, le=10)
    relationship_impact: conint(ge=-5, le=5)
    partner_reaction: Optional[str] = None
    days_after_reflection: Optional[conint(ge=0, le=30)] = None
    would_repeat_preparation: Optional[conint(ge=0, le=10)] = None
    reflection_short: Optional[str] = None
```

```
# backend/cqox/emotion/api.py
@router.post("/episodes/{episode_id}/outcome")
def submit_outcome(
    episode_id: int,
    outcome: schemas.OutcomeCreate,
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    service.save_outcome(db, user.id, episode_id, outcome)
    return {"status": "ok"}
```

4.3 因果効果取得 API

```
# backend/cqox/emotion/schemas.py
from typing import List

class TreatmentEffectRead(BaseModel):
    treatment_key: str
    outcome_name: str
    ate: float
    ci_lower: float
    ci_upper: float
    n_treated: int
    n_control: int
    model_version: str


class TreatmentEffectList(BaseModel):
    effects: List[TreatmentEffectRead]
```

```
# backend/cqox/emotion/api.py
@router.get("/effects/me", response_model=schemas.TreatmentEffectList)
```

```python
def get_my_effects(
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
    effects = service.get_effects_for_user(db, user.id)
    return schemas.TreatmentEffectList(effects=effects)
```

---

5. フロントエンド（可視化メイン）

5.1 Episode 作成ページ（既存の Layer A/B/C）

ここは概ね実装済みとのことなので、**バックエンドに `POST /episodes/draft` を飛ばす部分**だけ。

```tsx
// frontend/src/pages/EmotionEpisodeCreatePage.tsx
import React, { useState } from "react";
import { useMutation } from "@tanstack/react-query";
import { EpisodeQuickSliders } from "@/features/emotion/components/EpisodeQuickSliders";
import { PreferenceSliders } from "@/features/emotion/components/PreferenceSliders";
import { PreparationSimulator } from "@/features/emotion/components/PreparationSimulator";

export const EmotionEpisodeCreatePage: React.FC = () => {
  const [pre, setPre] = useState({ preAnxiety: 5, preCryingRisk: 5, preSpeechBlockRisk: 5 });
  const [prefs, setPrefs] = useState({ relief: 5, expression: 5, relationship: 5 });
  const [prep, setPrep] = useState({
    journaling_10m: 0,
    three_messages: 0,
    breathing_4_7_8: 0,
    roleplay_self_qa: 0,
    safe_word_plan: 0,
  });

  const saveDraftMutation = useMutation({
    mutationFn: async () => {
      const payload = {
        scenario_type: "interview", // TODO: フォームから取得
        topic: "転職理由",
        scheduled_at: new Date().toISOString(),
        location: "online",
        pre_state: {
          pre_anxiety: pre.preAnxiety,
          pre_crying_risk: pre.preCryingRisk,
          pre_speech_block_risk: pre.preSpeechBlockRisk,
        },
        preparations_planned: prep,
        preference_weights_raw: prefs,
      };
      const res = await fetch("/api/emotion/episodes/draft", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(payload),
      });
      if (!res.ok) throw new Error("Failed to save draft");
      return res.json();
    },
  });

  return (
    <div className="grid grid-cols-1 lg:grid-cols-2 gap-8">
      <div className="space-y-8">
        <EpisodeQuickSliders
          preAnxiety={pre.preAnxiety}
          preCryingRisk={pre.preCryingRisk}
          preSpeechBlockRisk={pre.preSpeechBlockRisk}
          onChange={setPre}
        />
        <PreferenceSliders
          relief={prefs.relief}
          expression={prefs.expression}
```

```
          relationship={prefs.relationship}
          onChange={setPrefs}
        />
      </div>
      <div className="space-y-4">
        <PreparationSimulator prep={prep} onChange={setPrep} pre={pre} prefs={prefs} />
        <button
          className="w-full rounded-xl py-3 font-semibold bg-blue-600 text-white disabled:opacity-50"
          disabled={saveDraftMutation.isLoading}
          onClick={() => saveDraftMutation.mutate()}
        >
          今回の準備プランを保存する
        </button>
      </div>
    </div>
  );
};
```

5.2 「あなたのパターン」ダッシュボード

5.2.1 ATE フォレストプロット

```
// frontend/src/features/emotion/components/CausalEffectForestChart.tsx
import React from "react";
import {
  ResponsiveContainer,
  BarChart,
  Bar,
  XAxis,
  YAxis,
  Tooltip,
  ReferenceLine,
  ErrorBar,
} from "recharts";

export type EffectDatum = {
  treatment_key: string;
  outcome_name: string;
  ate: number;
  ci_lower: number;
  ci_upper: number;
};

type Props = {
  data: EffectDatum[];
  outcomeName: string;
};

export const CausalEffectForestChart: React.FC<Props> = ({ data, outcomeName }) => {
  const filtered = data.filter((d) => d.outcome_name === outcomeName);
  const chartData = filtered.map((d) => ({
    name: labelForTreatment(d.treatment_key),
    ate: d.ate,
    ci_lower: d.ci_lower,
    ci_upper: d.ci_upper,
    err_low: d.ate - d.ci_lower,
    err_high: d.ci_upper - d.ate,
  }));

  return (
    <div className="border rounded-2xl p-4 shadow-sm">
      <h3 className="text-sm font-semibold mb-2">
        {outcomeLabel(outcomeName)} に対する準備ごとの推定効果
      </h3>
      <div className="h-64">
        <ResponsiveContainer width="100%" height="100%">
          <BarChart
            data={chartData}
            layout="vertical"
```

```
            margin={{ top: 10, right: 30, left: 100, bottom: 10 }}
          >
            <XAxis type="number" />
            <YAxis type="category" dataKey="name" />
            <Tooltip />
            <ReferenceLine x={0} strokeDasharray="3 3" />
            <Bar dataKey="ate">
              <ErrorBar
                dataKey="ate"
                width={4}
                data={chartData.map((d) => ({
                  x: d.ate,
                  low: d.ate - d.err_low,
                  high: d.ate + d.err_high,
                }))}
              />
            </Bar>
          </BarChart>
        </ResponsiveContainer>
      </div>
      <p className="mt-2 text-xs text-gray-500">
        棒は推定平均効果、線は 95% 信頼区間です。0 をまたいでいる場合は効果が不確かであることを示します。
      </p>
    </div>
  );
};

function labelForTreatment(key: string): string {
  switch (key) {
    case "journaling_10m":
      return "10分の書き出し";
    case "three_messages":
      return "伝えたい3つのメッセージ";
    case "breathing_4_7_8":
      return "4-7-8呼吸法";
    case "roleplay_self_qa":
      return "自分でQ&Aロールプレイ";
    case "safe_word_plan":
      return "セーフワードを決める";
    default:
      return key;
  }
}

function outcomeLabel(name: string): string {
  switch (name) {
    case "crying_level":
      return "涙レベル";
    case "stress_after":
      return "終わった後の楽さ";
    case "expression_score":
      return "伝えられた感";
    case "relationship_impact":
      return "関係への影響";
    default:
      return name;
  }
}
```

5.2.2 時系列チャート

```
// frontend/src/features/emotion/components/EmotionTimelineChart.tsx
import React from "react";
import {
  ResponsiveContainer,
  LineChart,
  Line,
  XAxis,
```

```
  YAxis,
  Tooltip,
  Legend,
  ReferenceLine,
} from "recharts";

type EpisodeTimelinePoint = {
  episode_id: number;
  label: string; // "面接 #23" など
  crying_level: number;
  expression_score: number;
  relationship_impact: number;
};

type Props = {
  data: EpisodeTimelinePoint[];
};

export const EmotionTimelineChart: React.FC<Props> = ({ data }) => {
  return (
    <div className="border rounded-2xl p-4 shadow-sm">
      <h3 className="text-sm font-semibold mb-2">エピソードごとの推移</h3>
      <div className="h-72">
        <ResponsiveContainer width="100%" height="100%">
          <LineChart data={data} margin={{ top: 10, right: 30, left: 0, bottom: 20 }}>
            <XAxis dataKey="label" angle={-45} textAnchor="end" height={60} />
            <YAxis />
            <Tooltip />
            <Legend />
            <ReferenceLine y={0} strokeDasharray="3 3" />
            <Line type="monotone" dataKey="crying_level" name="涙レベル" dot={false} />
            <Line type="monotone" dataKey="expression_score" name="伝えられた感" dot={false} />
            <Line type="monotone" dataKey="relationship_impact" name="関係への影響" dot={false} />
          </LineChart>
        </ResponsiveContainer>
      </div>
    </div>
  );
};
```

5.2.3 ダッシュボードページ

```
// frontend/src/pages/EmotionDashboardPage.tsx
import React from "react";
import { useQuery } from "@tanstack/react-query";
import { CausalEffectForestChart } from "@/features/emotion/components/CausalEffectForestChart";
import { EmotionTimelineChart } from "@/features/emotion/components/EmotionTimelineChart";

export const EmotionDashboardPage: React.FC = () => {
  const effectsQuery = useQuery({
    queryKey: ["emotion", "effects"],
    queryFn: async () => {
      const res = await fetch("/api/emotion/effects/me");
      if (!res.ok) throw new Error("failed to load effects");
      return res.json();
    },
  });

  const timelineQuery = useQuery({
    queryKey: ["emotion", "timeline"],
    queryFn: async () => {
      const res = await fetch("/api/emotion/episodes/timeline/me");
      if (!res.ok) throw new Error("failed to load timeline");
      return res.json();
    },
  });

  if (effectsQuery.isLoading || timelineQuery.isLoading) {
```

```
    return <div>読み込み中...</div>;
  }

  const effects = effectsQuery.data.effects;
  const timelineData = timelineQuery.data.points;

  return (
    <div className="space-y-8">
      <h1 className="text-xl font-semibold">あなたのパターン</h1>
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <CausalEffectForestChart data={effects} outcomeName="crying_level" />
        <CausalEffectForestChart data={effects} outcomeName="expression_score" />
      </div>
      <EmotionTimelineChart data={timelineData} />
    </div>
  );
};
```

---

6. 実行・検証フロー

6.1 環境構築と起動

```
# backend
cd backend
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt

# マイグレーション
alembic upgrade head

# API 起動
uvicorn cqox.api.main:app --reload --port 8000


# frontend
cd ../frontend
npm install
npm run dev -- --port 5173
```

6.2 最低限のテスト・検証

1. **ユニットテスト**

```
cd backend
pytest tests/emotion -q
```

2. **因果バッチ**

- 合成データ 5000 行を流し込んでから

```
python -m cqox.jobs.estimate_effects
psql -c "SELECT * FROM emotion_treatment_effect LIMIT 10;"
```

3. **フロント**

- `/emotion/create` で Episode を作り
- `/emotion/dashboard` で ATE と時系列が表示されることを確認

What this does（この回答・コードがしていること）

- **Goal（狙い）**
  Emotion CQOx を「**単なるシミュレーター付きフォーム**」から、
  - エピソードログ
  - 因果推論バッチ
  - 個人向け因果ダッシュボード

    - 準備プランのレコメンド
      を一貫して回せる **本番レベルの構成** に押し上げるための、
      アーキテクチャ＋DB＋因果ジョブ＋API＋フロント可視化のセットを示しました。
- **Inputs**
    - ユーザーが Episode ごとに入力する前後のスライダー値
    - 準備アクションの強度
    - ユーザーの優先度（楽さ / 伝えたい / 関係維持）
- **Outputs / side-effects**
    - PostgreSQL に Episode / Preparation / Outcome / Preference / TreatmentEffect が蓄積されます。
    - バッチ `estimate_effects.py` によって、ユーザー毎・準備毎・アウトカム毎の ATE と CI が推定され、`emotion_treatment_effect` に保存されます。
    - フロントの「あなたのパターン」ダッシュボードで、
        - 準備ごとの ATE フォレストプロット
        - エピソードごとの時系列推移
          が Recharts を使って可視化されます。
    - これにより、「この準備があなたの涙レベル / 楽さ / 伝えられた感にどのくらい効いてきたか」を具体的な数字・グラフで返せるようになります。
- **Why this approach**
    - 因果推論を「裏側のブラックボックスなモデル」ではなく、
        - 明示的な treatment / outcome / confounder
        - per-user ATE 推定
        - ATE をそのまま UI で可視化
          という形に落としているので、**どこに因果が使われているかが透明**。
    - モデル部分は DML 風の実装にしてあるので、
        - 将来 DoWhy/EconML/CausalForest 等への差し替えが容易。
    - 可視化は Recharts で完結させつつ、
        - フォレスト・時系列・カード UI を揃えて「分析者視点」でも耐えられる構成。
- **How to run / verify**
    1. 合成 CSV から Episode / Preparation / Outcome をインポート。
    2. `python -m cqox.jobs.estimate_effects` を実行し、`emotion_treatment_effect` に ATE が入るか確認。
    3. フロントの `/emotion/dashboard` を開き、
        - ATE が 0 付近ではなく、サンプル CSV の生成ロジックに対応した符号・大きさになっているかを見る。
    4. 新しい Episode を数件入力し、再度バッチを回して ATE の変化を確認する。

Expert insight (Google/Meta/NASA level)

因果推論を“プロダクト内ロジック”として本番投入する時に一番重要なのは、「どの数字が因果で、どの数字がそうでないか」を UI とスキーマの両方でラベル付けしておくことです。

- ここでは `emotion_treatment_effect` テーブルと、そのフォレストチャートだけが「ATE（do(T)) に対応する数字」で、
  それ以外（時系列・単純平均）はあくまで記述統計として扱うように切り分けています。
- この線を引いておけば、
    - どこまで「涙の原因」と言い切ってよいか
    - どこから先は「傾向・仮説」として提示すべきか
      をチーム内でも共有でき、**世界級プロダクトに必須の“誠実さと透明性”** を維持できます。