

Question②(Knowledge)



Thought for 10s >

了解。3点の指摘にまず短く回答→その後、**完全版 仕様+設計+ファイル配置**を“拡張なしの標準機能”として再提示します。

先に質問への回答

1. Grafana 37パネル

「例」ではなく**実数の“37”**です。項目は下の「#8 可観測性ダッシュボード（37枚・名称一覧）」に**全件列挙**しました。ダッシュボード JSON は `grafana/dashboards/cqox_integrated.json`（1枚に37パネル）と `grafana/dashboards/*.json`（用途別）に配置します。

2. ドメインは“全対応がデフォルト”

プリセット削除。**完全ドメイン非依存**が基本です。推定・可視化・品質ゲートは共通、列名辞書・語彙オントロジ・自動ロール推定器を**標準**で同梱（`config/ontology/*`）。「医療」「教育」などの言葉は UI 文言と例だけにとどめます（挙動差分なし）。

3. 図表（DiD 等10+）は“このプロダクトに内蔵”

バックエンドで**PNG/HTML生成**と**Base64返却**、フロントは**描画（Recharts/Plotly）**の両方を**標準実装**。ファイル保存は `reports/figures/<job_id>/...`、フロント実装は `frontend/src/components/figures/*`。図名は下の「#6.4 図表セット（14種）」に全件明記しています。

CQO-COMPLETE : 最終版 仕様 + 設計（拡張なし = 全部標準）

1. 目的

- 観測データから因果効果を**本番運用品質**で出す。
- 7推定器×7品質ゲート**の合議 + **Fail-Closed**（1つでもFailなら出力拒否）。
- GPU加速**・非同期ストリーミング・可観測性（**SLO管理**）まで**一体**の製品。

2. 全体構成（ディレクトリも確定）

```
cqox-complete/
  backend/
    gateway/          # FastAPI REST : アップロード/ロール推定/API集約
    engine/           # 推定器オーケストレーション/品質ゲート/CAS
    estimators/       # tvce, ope, hidden, iv, transport, proximal, network
    quality_gates/    # ess, tail, ci_width, weak_iv, sensitivity, balance, mono
    grpc_common/      # .proto, stubs
    workers/gpu/      # gRPC Worker (HMC/JAX, cuPy/Numba)
    observability/    # Prometheus export, logging, tracing
  frontend/
    src/
      lib/client.ts   # APIクライアント (/api/* & WS)
      components/
        figures/      # 図表14種のUI (Plotly/Recharts)
        panels/       # QualityBoard, CAS Radar, Heterogeneity 等
        pages/        # Upload, Roles, Configure, Run, Results, Dashboard
  config/
    ontology/
      columns.json    # 列名シノニム辞書 (全ドメイン)
      units.json      # 単位/尺度辞書 (% , rate, log等)
      validators.json # 型/範囲/日付フォーマット規則
  data/uploads/       # CSV保存
  reports/figures/    # バックエンドが書き出すPNG/HTML (job_id別)
  grafana/dashboards/ # JSON (統合1枚=37パネル)
  prometheus/         # prometheus.yml (targets: gateway, engine, worker)
  docker/             # docker-compose.yml, Dockerfile.*
  k8s/               # Deployment/Service/HPA/ServiceMonitor 等
```

3. API (REST / WS / gRPC)

3.1 REST (Gateway)

- ``POST /api/upload` → `{ ok, dataset_id, meta:{columns, rows, preview_rows[] } }``
- ``GET /api/roles/profile?dataset_id=...` → 列の型/候補ロール/信頼度`
- ``POST /api/analyze/comprehensive``

```
{
  "dataset_id": "...",
  "mapping": { "y": "y", "w": "treatment", "unit_id": "user_id", "time": "date" },
  "cfg_json": { "lambda": 1.0, "sample_limit": 10000, "seed": 123 },
  "preview": true
}
```

- ``GET /api/jobs/:job_id/status`` / ``GET /api/jobs/:job_id/report`` (JSON+図表リンク)

3.2 WebSocket (UIプレビュー)

- ``GET /ws/stream/:job_id``
進捗, gate通過率, 中間サンプル, CAS推移, ログ要約

3.3 gRPC (Engine⇄GPU Worker)

- ``PullTasks(stream Hello) -> stream WorkRequest``
- ``SubmitResult(stream WorkResponse) -> Ack``
``WorkRequest` : `job_id, kind, df_path, mapping, cfg_json, seed``
``WorkResponse` : `progress, partial, final, metrics, error?``

4. 推定器 (7) と品質ゲート (7)

4.1 推定器 (同時実行)

- **tvce** (時変DID / イベントスタディ / 並行トレンド検定)
- **ope** (IPW/DR、安定化重み、トリミング)
- **hidden** (Rosenbaum感度曲線)
- **iv** (2SLS、First-stage F>10 まで自動IV探索)
- **transport** (Target shift への再重み付け)
- **proximal** (負のコントロール利用)
- **network** (スピルオーバー／グラフ干渉)

4.2 品質ゲート (Fail-Closed)

- **ess** (有効サンプル)
- **tail** (重みの裾・外れ率)
- **ci_width** (CI幅しきい値)
- **weak_iv** (Stock-Yogo 10超)
- **sensitivity** (Rosenbaum Γ)
- **balance** (SMD<0.10)
- **mono** (単調性・シンプソン回避)

4.3 CAS (Causal Assurance Score, 0-100)

``CAS = w1*GatePass + w2*SignConsensus + w3*CIOverlap + w4*DataHealth + w5*Sensitivity``

5. ドメイン完全非依存の仕掛け (辞書・自動ロール)

- 列名辞書 ``config/ontology/columns.json``

```
{
  "treatment": ["treat", "variant", "policy", "exposure", "dose"],
  "outcome":   ["y", "outcome", "label", "metric", "score", "KPI", "conversion"],
  "unit_id":   ["id", "user_id", "patient_id", "device", "account", "node"],
  "time":      ["date", "timestamp", "period", "week", "day", "t", "time"],
  "cost":      ["cost", "price", "spend", "loss"],
  "log_propensity": ["log_propensity", "lp", "logit_p", "pscore_logit"],
}
```

```
"cluster_id": ["cluster", "network", "class", "group", "cohort"]
}
```

- 型/範囲ルール `validators.json` (日時推定, 比率/確率, ログ指標検出 等)
- 自動ルール推定: 列名一致 + 分布 + 相関 + テキスト埋め込みの複合スコアで `y,w,unit_id,time,cost,log_propensity,cluster_id` を推定。信頼度と候補を UI に提示。
- どの業種でもこの辞書・検出器で同じパイプラインが走る (挙動差分なし)。

6. UI (画面/コンポーネント/図表)

6.1 画面

1. **Upload**: CSV投入、即時プレビュー (先頭N行/型/欠損/外れ)
2. **Roles**: 自動推定のルールを確認・修正 (信頼度バー)
3. **Configure**: λ 、サンプル制限、Heterogeneity軸、Network設定
4. **Run**: 進捗、realtimeプレビュー (WS)、ゲート通過率
5. **Results**: 7推定器の効果/CI、合議、**QualityBoard**、**CAS Radar**
6. **Decision Dashboard**: 推奨、リスク、改善ガイド、自動レポートDL

6.2 主要コンポーネント

- `QualityBoardPanel` (7ゲートのPass/Failヒートボード)
- `CASRadar` (CAS 5成分のレーダー)
- `EstimatorTable` (効果/CI/時間/符号一致)
- `HeterogeneityView` (サブグループ差)
- `NetworkSpilloverMap` (干渉ヒートマップ/グラフ)
- `AdviceCard` (Fail時の具体的アクション)

6.3 技術

- Plotly** (イベントスタディ係数、感度曲線、レーダー)
- Recharts** (分布/棒/帯)
- 大規模表は仮想化 (100k行でも滑らか)
- 状態は `queryキー=job_id` でURL共有可能

6.4 図表セット (14種 = 常時)

1. **Parallel Trends (DID 前提)**
2. **Event Study Coefficients** (\pm 期間)
3. **ATE Density** (推定器別)
4. **Propensity Overlap Histogram**
5. **Balance SMD Lollipop**
6. **Rosenbaum Sensitivity Curve**
7. **IV First-Stage F** 分布/しきい
8. **IV 強度 vs 2SLS 効果の安定性**
9. **Transport Weights Distribution**
10. **Time-Varying Effect Line (TVCE)**
11. **Network Spillover Heatmap/Adj**
12. **Heterogeneity Waterfall** (上位10群)
13. **Quality Gates Board** (7枚の合否, 1枚統合)
14. **CAS Radar** (5軸)

保存場所: `reports/figures/<job_id>/{parallel_trends.png, event_study.html, ...}`

UI実装: `frontend/src/components/figures/*` (それぞれ1ファイル)

7. 出力 (最終JSON + 図リンク)

- `engine` が最終JSONを返す: `cas, gates{}, estimators[], figures{key->url/base64}, advice[]`
- UI は JSON をそのまま保存/DL、図は埋め込み or ローカルリンクを選択可能。

8. 可観測性ダッシュボード (37パネル：名称一覧)

1. Engine E2E Latency p50/p95/p99
2. Gateway Request Latency p50/p95
3. Upload Throughput RPS
4. Analyze Throughput RPS
5. Error Rate (4xx/5xx)
6. Job Queue Depth (engine)
7. Worker Concurrency (実行中ジョブ)
8. Worker Task Wait Time
9. GPU Memory Used %
10. GPU Utilization %
11. GPU OOM/Timeout Count
12. Estimator Latency (tvce)
13. Estimator Latency (ope)
14. Estimator Latency (hidden)
15. Estimator Latency (iv)
16. Estimator Latency (transport)
17. Estimator Latency (proximal)
18. Estimator Latency (network)
19. Gate Pass Rate (ess)
20. Gate Pass Rate (tail)
21. Gate Pass Rate (ci_width)
22. Gate Pass Rate (weak_iv)
23. Gate Pass Rate (sensitivity)
24. Gate Pass Rate (balance)
25. Gate Pass Rate (mono)
26. **CAS Average** (全ジョブ)
27. **CAS Distribution Histogram**
28. Sign Consensus Ratio (+一致率)
29. CI Overlap Index
30. Data Health (missing+imbalance 指標)
31. Reject (Fail-Closed) Rate
32. Domain-Agnostic Mix (自動判定カテゴリ比)
33. Largest File Size Processed (rolling)
34. P95 Time per 10k Rows
35. Uptime / Availability
36. Top Error Reasons (分類)
37. End-to-End SLO Compliance Heatmap

収集元：`/observability/metrics.py` (Prometheus)

ダッシュボード：`grafana/dashboards/cqox_integrated.json` (この1枚に37パネル)

9. セキュリティ/監査 (標準)

- PII自動マスク (列検出→ハッシュ)
 - RBAC (Viewer/Analyst/Admin)
 - 監査ログ (操作/設定/コードハッシュ/乱数シード/環境)
 - 署名つきレポート (エクスポートJSON/PDF)
-

10. デプロイ (要点)

- 単体：`uvicorn backend.gateway.app:app -p 8081` / `uvicorn backend.engine.server:app -p 8080` / `python workers/gpu/worker.py`
- **Compose**：`docker/docker-compose.yml` (gateway, engine, worker-gpu, prometheus, grafana)

- **K8s** : ``k8s/`` (Deployment, HPA, ServiceMonitor, Grafana provisioning)

11. ランブック (例)

- UI 404 ``/api/upload`` → Gateway未起動 or Vite proxy (``target: http://localhost:8081``)
- ``422 Unprocessable`` → ``mapping`` 不整合。Roles で修正→再送
- CAS<50 → QualityBoard Failのカードに従いデータ改善 (ESS↑、IV再選定、λ↑)

まとめ

- **37**パネルは実数で全件定義。
- ドメイン完全非依存 (辞書/検出器は標準、挙動差分なし)。
- 図表**14**種は内蔵し、ファイル保存 + UI描画の両対応。
- 本回答の名称・配置・APIでそのまま実装/レビュー/運用に移せます。