# TrustBench (aka AegisEval) — System Diagrams

This page includes multiple Mermaid diagrams you can copy into docs, READMEs, or wikis. They cover: (1) System Context (C4 L1), (2) Platform Components (C4 L2), (3) CI Flow, and (4) Studio App User Flow.

---

## 1) System Context (C4 Level 1)

```
flowchart LR
  actorUser["End Users / Students"]
  actorEng["Engineers / Contributors"]
  actorSec["Security / Red Team"]
  actorPM["PM / Compliance / Audit"]

  subgraph TrustBench[TrustBench Platform]
    core["trustbench-core\n(eval runners, judges, metrics)"]
    ci["trustbench-ci\n(GitHub Action / CLI)"]
    studio["trustbench-studio\n(Web App / Dashboard)"]
    datasets["trustbench-datasets\n(golden sets, red-team suites)"]
    reports["trustbench-reports\n(JSON/CSV/HTML artifacts)"]
  end

  repo["GitHub Repos\n(agent code, prompts, configs)"]

  actorEng -->|PRs, pushes| repo
  actorUser -->|Upload/Paste repo| studio
  actorSec -->|Schedule adversarial runs| studio
  actorPM -->|Review metrics & artifacts| studio

  ci --> core
  studio --> core
  core --> reports
  core --> datasets
  core -->|clone| repo

  reports --> actorPM
  reports --> actorSec
  reports --> actorEng
  datasets --> core
```

---

## 2) Platform Components (C4 Level 2 — Containers)

```
flowchart TB
  subgraph Core[trustbench-core]
    runner["Eval Runner\n(orchestrates pillars & tools)"]
    judge["LLM-as-Judge\n+ HITL calibration"]
    pillars["Pillars:\n1) Task Fidelity\n2) System Robustness\n3) Security\n4)
Ethics/Refusal"]
    metrics["Metrics Engine\n(faithfulness, injection-block rate, p95 latency,
etc.)"]
  end

  subgraph CI[trustbench-ci]
    gha["GitHub Action Wrapper"]
    cli["CLI Adapter (local runs)"]
  end

  subgraph Studio[trustbench-studio]
    ui["Web UI (Streamlit/Next.js)"]
    api["/evaluate API (job queue)"]
    store["Run Store (SQLite/Postgres/FS)"]
  end

  subgraph Datasets[trustbench-datasets]
    gold["Golden Examples"]
    red["Red-Team Suites"]
  end

  subgraph Reports[trustbench-reports]
    json["JSON / CSV / HTML artifacts"]
    dash["Visual Dashboard"]
  end

  Repo["GitHub Repo (agent code)"]

  gha --> runner
  cli --> runner
  ui --> api --> runner
  runner --> judge
  runner --> pillars
  runner --> metrics
  runner --> json
  json --> dash
  runner -->|clone/read| Repo
  gold --> runner
```

```
  red --> runner
  dash --> store
```

---

## 3) CI Flow — From Push to Gate

```
sequenceDiagram
  participant Dev as Developer
  participant GH as GitHub (PR)
  participant CI as trustbench-ci (Action)
  participant Core as trustbench-core (Runner)
  participant Rep as Reports

  Dev->>GH: Push branch / Open PR
  GH->>CI: Trigger workflow (eval.yaml)
  CI->>Core: Run evaluation (clone repo, detect entrypoints)
  Core->>Core: Execute pillars (Task/System/Security/Ethics)
  Core->>Rep: Save artifacts (JSON/CSV/HTML + failures)
  Core-->>CI: Return metrics + pass/fail per-threshold
  CI-->>GH: PR summary comment + artifacts link
  alt Any pillar below threshold
    GH-->>Dev: ❌ Block merge (required check fails)
  else All thresholds met
    GH-->>Dev: ✅ Merge allowed
  end
```

---

## 4) TrustBench Studio — User Flow

```
flowchart LR
  U[User (student/learner)] --> A[Paste GitHub URL or Upload Folder]
  A --> B{Profile Selected?}
  B -- default --> C[Use default thresholds]
  B -- custom --> D[Select profile & thresholds]
  C --> E[Start Evaluation]
  D --> E[Start Evaluation]
  E --> F[Progress + Live Logs]
  F --> G[Results Dashboard\n(scorecards per pillar)]
  G --> H[Inspect Failure Artifacts]
  H --> I[Suggested Fixes (retriever, prompt, guard)]
  I --> J[Re-run Evaluation]
  G --> K[Export Report (JSON/HTML/PDF)]
```

---

## Notes & Legend

- **Pillars**: Task, System, Security, Ethics
- **Artifacts**: exact failing I/O pairs, judge rationale, timestamps, seeds
- **Thresholds**: per-profile, used to gate merges in CI
- **HITL**: human labels calibrate judge prompts and thresholds

---

---

## 5) Skill-Based Agents with MCP — Patterned Topology (avoids common antipatterns)

```
flowchart LR
  %% High-level orchestrator with explicit skills to avoid "one agent to rule
them all"
  Start([__start__]) --> Orchestrator{{Router/Planner}}
  subgraph MCP[Model Context Protocol Layer]
    GH[(GitHub Server)]
    RepoScan([scan_repo])
    Secrets([secrets_scan])
    Semgrep([semgrep_rules])
    VT([vt_lookup])
    KB([/api/reports/latest])
    RAGAS([ragas_eval])
    PromptGuard([prompt_guard])
  end

  %% Branch into skill-based agents (each with a clear pattern)
  Orchestrator -->|route by profile| TaskFidelity
  Orchestrator -->|route| SecurityEval
  Orchestrator -->|route| SystemPerf
  Orchestrator -->|route| EthicsRefusal

  %% Agent nodes labeled by best-suited patterns
  TaskFidelity["Task Fidelity Agent
Pattern: **Planner→Executor** with **Retrieval Router**
Skills: chunk, embed, retrieve, compare vs truth"]
  SecurityEval["Security Red-Team Agent
Pattern: **Adversary→Defender Duel** + **Critic/Referee**
Skills: jailbreak gen, injection tests, guard updates"]
  SystemPerf["System Performance Agent
Pattern: **Sampler→Aggregator**
Skills: latency sampling, p95 calc, stability checks"]
  EthicsRefusal["Ethics/Refusal Agent
Pattern: **Critic/Referee**
```

```
Skills: policy check, refusal accuracy, content filters"]

  %% Tools via MCP (explicit edges to avoid blurry boundaries)
  TaskFidelity -->|use| RAGAS
  TaskFidelity -->|fetch| GH
  SecurityEval -->|attack/score| PromptGuard
  SecurityEval -->|code scan| Semgrep
  SecurityEval -->|secrets| Secrets
  SecurityEval -->|hash intel| VT
  SystemPerf -->|logs| KB
  EthicsRefusal -->|policies| KB

  %% Aggregation and gating
  TaskFidelity --> Agg((Aggregation))
  SecurityEval --> Agg
  SystemPerf --> Agg
  EthicsRefusal --> Agg

  Agg --> Scores["Score Synthesizer
(faithfulness, injection-block, refusal_acc, p95)"]
  Scores --> Report["Reports (JSON/CSV/HTML)"]
  Scores --> Gate{CI Gate}
  Gate -->|pass| End([__end__])
  Gate -->|fail| PRComment["PR Comment + Artifacts"]

  %% Anti-pattern guards (annotations)
  classDef anti fill:#222,stroke:#f66,color:#f66;
  note1[[Avoid: One Agent To Rule Them All]]:::anti
  note2[[Avoid: Death by 1000 Agents]]:::anti
  Orchestrator --- note1
  Agg --- note2
```

**Why this avoids antipatterns:** - **Overloaded agent** → split into four *skill* agents with explicit responsibilities. - **Too many tiny agents** → one agent per *pillar*, not per micro-step; internal tools remain tools, not agents. - **LLM hammer** → non-LLM skills (Semgrep, secrets scan, p95 sampling) are tools via MCP, not prompts. - **Chain of pain** → branches run in parallel; aggregation is single, testable node. - **Blurred boundaries** → each agent lists its skills and tools; MCP edges make dependencies explicit. - **No escape hatch** → CI gate provides fail-safe; artifacts enable human review and retry.

---

## 6) MCP Handshake & Tool Use (sequence)

```
sequenceDiagram
  participant Studio as TrustBench Studio (UI)
  participant Core as Orchestrator (LangGraph)
  participant MCP as MCP Client
```

```
  participant GH as GitHub Server
  participant Tools as Tools (semgrep/secrets/VT/prompt_guard/ragas)

  Studio->>Core: /evaluate(repo_url, profile)
  Core->>MCP: discover()
  MCP-->>Core: tools=[scan_repo, secrets_scan, semgrep_rules, vt_lookup,
ragas_eval, prompt_guard]
  Core->>GH: clone(repo)
  Core->>MCP: call(scan_repo, path)
  Core->>MCP: call(ragas_eval, golden_set)
  Core->>MCP: call(secrets_scan | semgrep_rules | vt_lookup)
  Core->>MCP: call(prompt_guard, prompts)
  Core-->>Studio: metrics + failing I/O + rationale
  Studio-->>Studio: visualize & gate (if CI)
```

---

## 7) Pillar Mini-Graphs (like your tag-extractor screenshot)

```
flowchart TB
  subgraph Security_Pillar
    start([__start__]) --> jailbreak[generate_attacks]
    start --> secrets[scan_secrets]
    start --> static[semgrep_scan]
    jailbreak --> defense[prompt_guard_score]
    secrets --> aggS
    static --> aggS
    defense --> aggS[aggregate_security]
    aggS --> end([__end__])
  end

  subgraph Task_Pillar
    tstart([__start__]) --> retrieve[retrieve_topk]
    tstart --> baseline[extractive_fallback]
    retrieve --> compare[ragas_faithfulness]
    baseline --> compare
    compare --> tend([__end__])
  end
```

---

### Next Steps

- Confirm final agent names and the exact MCP tool list
- I can export these diagrams to a downloadable `.md` or embed them into your README