Policy-Driven AI Guardrail Engine

Goal

Build a **deterministic guardrail layer** that decides what to do with AI outputs based on **policy configuration**.

This layer sits **between an AI system and end users**.

Allowed

- Internet
- AI tools
- Python

Inputs

`policies.json`

Defines guardrail rules.

Each policy:

- applies to a `risk` type
- restricts what actions are allowed
- enforces a minimum confidence

```
{
  "policies": [
    {
      "id": "P1",
      "risk": "medical",
      "allowed_actions": ["escalate"],
      "min_confidence": 0.95
    },
```

```
    {
      "id": "P2",
      "risk": "financial",
      "allowed_actions": ["sanitize", "escalate"],
      "min_confidence": 0.85
    },
    {
      "id": "P3",
      "risk": "general",
      "allowed_actions": ["allow"],
      "min_confidence": 0.7
    }
  ],
  "default_action": "block"
}
```

Interpretation notes:

- Policies are **data**, not code
- More than one policy may apply to the same input
- Policies may be incomplete or malformed → handle safely

---

inputs.json

Represents AI-generated responses with metadata.

```
[
  {
    "id": "R1",
    "risk": "medical",
    "output": "You should take this medicine daily",
    "confidence": 0.88
  },
  {
    "id": "R2",
    "risk": "general",
    "output": "You can reset your password from settings",
```

```
    "confidence": 0.92
  }
]
```

Interpretation notes:

- `confidence` is given by the AI system
- You are **not** judging correctness of text
- You are deciding **whether it is safe to act on it**

Task

For each input:

1. Find all applicable policies
2. Evaluate confidence thresholds
3. Choose a **single final action**
4. Explain the decision

Actions

Exactly one per input:

- `allow` → show output as-is
- `sanitize` → replace output with safe fallback
- `escalate` → human review
- `block` → suppress output

Multi-Policy Matching (Important)

- Multiple policies may match the same input
- You must apply the **most restrictive outcome**

Examples of “more restrictive” (not exhaustive):

- `block` > `escalate` > `sanitize` > `allow`
- policy with higher safety risk overrides lower
- unmet confidence threshold increases restriction

You must **define and implement** this logic.

Sanitization

If action = `sanitize`:

- do **not** partially edit the original text
- replace it with a generic safe response

Example:

> “This response cannot be shown. Please consult a qualified professional.”

Exact wording is up to you.

Output

Write `output.json`

```
{
  "id": "R1",
  "decision": "escalate",
  "applied_policies": ["P1"],
  "final_output": "Sent for human review",
  "reason": "medical risk; confidence 0.88 < required 0.95"
}
```

Requirements:

- decision must be explainable from data

- explanation must reference policy logic

Engineering Expectations

- Python only
- Deterministic (no randomness)
- Policy logic not hardcoded
- Clear separation:
    - policy loading
    - matching
    - decision resolution
- Safe handling of bad input

Submission (Mandatory)

- Create a **public GitHub repository**
- Commit code during the exercise
- Repository must include:
    - source code
    - input files
    - generated `output.json`
    - `README.md`:
        - how to run
        - assumptions
        - tradeoffs

Commit history is reviewed.

Bonus

- Unit tests
- Packaging
- Focus on:

    - multi-policy conflicts

    - confidence thresholds
    - default behavior

Optional (Pick One)

- Audit mode: show all matched policies
- Rule trace: show which checks passed/failed
- CLI flags for file paths