Healing and Retries

This document describes how the system enforces correctness, termination, and safety when interacting with LLM-generated artifacts. Healing and retries are not “quality improvements”; they are **containment mechanisms** that exist to preserve system invariants when the LLM inevitably produces invalid, inconsistent, or unsafe outputs.

---

1. System Invariants (Non-Negotiable)

All healing and retry logic exists **only** to restore these invariants. If they cannot be restored, the system halts.

1.1 Bounded Execution

The system **must terminate**.

- No infinite loops
- No unbounded retries
- No unbounded token usage
- All execution guarded by hard timeouts

1.2 Fail-Closed Correctness

The system **never fabricates correctness**.

- Incorrect outputs are never returned
- Unknown outputs are never guessed
- Partial correctness is preferred over speculative completeness

1.3 Local Verifiability

Every accepted artifact must:

- Execute locally (no LLM required)

- Produce deterministic output
- Be reproducible under the same inputs

1.4 Separation of Responsibility

- **LLM proposes**
- **Platform verifies**
- **Platform decides**

The LLM is never trusted as an authority—only as a candidate generator.

---

2. Oracle (Test Generation) Retry — Stage 4

2.1 What Is Being Retried?

The **entire test-generation program** produced by the LLM:

- `generate_inputs()`
- `is_valid_input(*args)`

This is **not patching**—it is full regeneration.

2.2 Retry Conditions

A retry is triggered if **any** of the following occur:

Static Validation Failures

- Missing required functions
  (Must define *exactly* two functions with correct signatures)
- Forbidden imports (`import`, `from`, filesystem, network)
- Syntax errors

Runtime Failures

- Execution timeout (> 6s)
- Runtime crash
- Non-deterministic behavior

Semantic Failures

- Fewer than `MIN_EXECUTABLE = 10` valid inputs
- Inputs rejected by `is_valid_input`
- Excessive duplication (duplicate test tuples are not tolerated)

2.3 Retry Mechanics

Each retry:

- Includes the **previous broken generator**
- Injects **precise diagnostics** (counts, rejection reasons)
- Appends a **mandatory correction directive**

MAX_RETRIES = 3

Temperature Schedule

`0.5 → 0.7 → 0.9`

2.4 Termination Guarantee

After 3 failures:

- Stage 4 **halts**
- A `400` error is returned
- The system does **not** guess or degrade constraints

Stage 4 cannot loop infinitely.

---

3. Reference Solution Generation & Consensus — Stage 5

3.1 Dual-Solution Architecture

Two independent LLM-generated programs:

- **Brute-Force Solution**
    - Treated as *practical ground truth*
    - Optimized for correctness, not performance
- **Optimized Solution**
    - Intended final reference solution

This redundancy is intentional.

3.2 Consensus Protocol

For each executable test:

1. Run optimized solution (2s timeout)
2. Determine source of truth:
    - User-provided output (highest priority)
    - Else brute-force output
3. Compare outputs

A single mismatch triggers healing.

---

4. Self-Healing (Localized Program Repair)

4.1 Cause

Mismatch between optimized output and ground truth.

4.2 Healing Strategy

This is **program repair**, not re-synthesis.

For each failing input:

- Construct a targeted prompt containing:
    - Failing input
    - Optimized output
    - Expected output
    - Source of truth (User vs Brute)

The LLM is instructed to **repair the logic**, not rewrite from scratch.

MAX_HEALS = 3
Temperature = 0.1

4.3 Circuit-Breaker (Hard Stop)

If `heals >= MAX_HEALS`, the system halts healing.

At this point, one or more of the following is assumed:

- Brute-force solution is wrong
- Test suite is contradictory
- Specification is inconsistent
- Problem is underspecified
- The LLM cannot represent the solution

Continuing would violate bounded execution.

Decision:
The last optimized solution is returned **as-is**, with no further claims.

---

5. Gold Test Suite Acceptance Policy

Each test is executed independently in a sandboxed process.

Failure Modes

- Timeout → drop test
- Runtime error → drop test
- Serialization error → drop test
- Success → accept test

Let:

```
failure_rate = 1 - (valid_tests / total_tests)
```

- If `failure_rate > 30%` → **Reject test suite**
- Else → Accept partial suite

Rationale: real test generation contains edge-case noise.

---

6. Circular Healing (Tests ←→ Solutions)

If the **reference solution rejects too many tests**:

- The system blames the **test generator**
- Feedback is injected
- Stage 4 regenerates tests
- Stage 5 re-runs consensus

This mirrors property-based testing systems:

> If the oracle fails too often, the generator is wrong.

---

7. User-Driven Healing (Ground Truth Injection)

When users submit counterexamples:

1. Gatekeeper validates constraint compliance
2. Valid inputs become **absolute truth**
3. These override brute-force outputs
4. Stage 5 healing reruns with injected truth

Important:
Even user truth does **not** bypass:

- `MAX_HEALS`
- Termination guarantees

The system always halts.