End-to-End Test Case v1 — Heterogeneous Distributed Inference with Fuzzy Verification

Purpose: Design a single, concrete, end-to-end test case that validates the *core technical claims* of the system: - heterogeneous hardware participation - pipeline-parallel inference - fuzzy (canonical-grid) verification - optimistic audits - fault tolerance under node failure

This test case is intentionally scoped so it can be implemented and executed by **one developer** using: - **MacBook Air (Apple M4, 24 GB RAM)** - **Desktop PC (RTX 4070 Super)**

The goal is to provide a specification detailed enough that an **AI Coding Planning Agent** could: 1. derive a concrete implementation plan, 2. write the required services and scripts, 3. run and validate the experiment.

---

1) Test Objective

Demonstrate that: 1. Partial inference can be executed across heterogeneous devices. 2. Canonical-grid commitments prevent false slashing despite hardware differences. 3. Random audits correctly accept honest computation. 4. The system recovers cleanly from a mid-pipeline node failure.

Success means **correct output**, **no false fraud detection**, and **session completion despite failure**.

---

2) Hardware & Environment

2.1 Physical Nodes

Node ID	Machine	Hardware	Role
N0	Desktop	RTX 4070 Super (CUDA)	Coordinator + Compute
N1	Desktop	RTX 4070 Super (CUDA)	Compute Worker
N2	MacBook Air	Apple M4 (Metal/MPS)	Compute Worker + Verifier

Coordinator runs on N0 for simplicity.

2.2 Software Stack (Strict)

Backend: - `llama.cpp` - Quantization: `GGUF Q4_K_M`

Model: - `Llama-3-8B-Instruct` (or equivalent open-weight 8B model)

Runtime Constraints: - Fixed llama.cpp version (pinned commit hash) - Deterministic mode enabled - Fixed thread counts per node - Canonicalizer module enabled and mandatory

Transport: - gRPC over TCP (localhost / LAN)

---

3) Model Partitioning (Pipeline Parallelism)

3.1 Layer Assignment

Assume Llama-3-8B has 32 transformer layers.

Node	Layers	Notes
N1	Layers 0–10	Early layers (prompt sensitive)
N2	Layers 11–21	Mid layers (untrusted OK)
N0	Layers 22–31	Final layers + logits

Rationale: - MacBook handles mid layers only. - Desktop handles early + late layers.

---

4) Canonical Grid Commitment (Verification Projection)

4.1 Canonicalization Function (Fixed)

Applied **only for verification**, not for forward inference.

Steps: 1. Cast output tensor to `float16` 2. Grid snap:

```
y_grid = round(y * 64) / 64
```

3. Clamp values:

```
y_clamped = clip(y_grid, -100.0, 100.0)
```

4. Serialize as little-endian bytes 5. Hash using SHA-256

This hash is the **output commitment**.

---

5) Test Workflow (Happy Path)

5.1 Session Initialization

Coordinator (N0) creates `InferenceSession`: - model_id: `llama3-8b-q4_k_m` - rng_seed: fixed - audit_probability: 0.2 (20%)

Placement: - Stage 0 → N1 - Stage 1 → N2 - Stage 2 → N0

5.2 Token Generation Loop (Single Prompt)

Prompt:

```
Explain in one paragraph why the sky appears blue.
```

Target length: 64 tokens.

For each token: 1. N1 computes layers 0–10 - emits `h_10` - commits canonical hash 2. N2 computes layers 11–21 - emits `h_21` - commits canonical hash 3. N0 computes layers 22–31 - produces logits - coordinator samples token

Receipts stored for every stage.

---

6) Audit Procedure

6.1 Random Audit Selection

For ~20% of work units (random): - Coordinator selects N0 as verifier - Recompute the audited shard

6.2 Verification Rule

- Recompute output
- Apply canonicalization
- Compare hash with worker commitment

Expected result: - **All audited units pass** despite different hardware backends.

Failure criteria: - Any false mismatch = test failure.

---

7) Fault Injection Test (Critical)

7.1 Failure Scenario

At token index = 20: - Forcefully terminate N2 (MacBook) worker process.

7.2 Expected Coordinator Behavior

1. Timeout triggers for stage 1
2. Coordinator selects backup: N0
3. Coordinator requests resend of `h_10` from N1
4. N0 computes layers 11–21 locally
5. Pipeline resumes

7.3 Acceptable Outcomes

- One-token latency spike
- No session restart
- Generation continues to completion

Unacceptable outcomes: - Session abort - Corrupted output - False fraud proof

---

8) KV-Cache Handling (v1 Constraint)

During failover: - N0 recomputes KV-cache for layers 11–21 from prompt - Performance degradation acceptable - Correctness required

Document observed latency impact.

---

9) Metrics to Capture

9.1 Correctness Metrics

- Final generated text matches single-node baseline (exact token match)
- No false fraud proofs

9.2 Verification Metrics

- Audit pass rate (expected ~100%)
- Hash stability across hardware

9.3 Performance Metrics

- Per-token latency per stage
- Latency spike during failover

• Throughput (tokens/sec)

---

10) Success Criteria (Binary)

The test is considered **successful** if:

1. Inference completes end-to-end
2. Output text is correct and coherent
3. All audits pass
4. Failover completes without session abort

---

11) Deliverables for the Coding Agent

The AI Coding Planning Agent should produce:

1. **Coordinator service**
2. session management
3. placement
4. audit logic
5. failover logic
6. **Worker daemon**
7. llama.cpp wrapper
8. PP execution
9. canonicalizer
10. receipt signing
11. **Verifier logic**
12. recomputation
13. hash comparison
14. **Test harness**
15. baseline single-node inference
16. distributed run
17. diff + report
18. **Runbook**

19. how to start nodes
20. how to inject failure
21. how to interpret results

---

12) Extension Tests (Optional, After Success)

- Increase audit rate to 50%
- Increase batch size to test activation bandwidth
- Introduce intentional faulty worker
- Vary grid size to test tolerance margins

End of Test Case v1