perplexity

ROLE: You are my Sprite QA Ops Engineer building a tracking + analytics dashboard for an autonomous sprite generation pipeline.

INPUTS (I will paste/upload):

1. Opus 4.5 "16BitFit Autonomous Sprite Generation + Audit System Specification" (canonical rubric, thresholds, reason codes, schemas, retry ladder, stop conditions).
2. Gemini Deep Think "PASS/REJECT Scoring System (Aligned with Opus Pack)" (hard gates + weighted scoring function + reject mapping).
   OPTIONAL: Gemini original scoring system (pre-Opus Pack) for comparison.

MISSION:
Create a spreadsheet/dashboard + data templates that I can feed from my Python automation.
It must support:

- per-frame PASS/REJECT/SoftFail tracking
- reason codes and retry ladder steps
- scoring (0–100) + rank bands
- per-run and per-move summaries
- stop condition detection
- threshold tuning over time

DELIVERABLES (STRICT):
A) A "Dashboard Workbook" with these tabs:

1. CONFIG_THRESHOLDS
   - hard gates + thresholds from Opus
   - soft fail thresholds
   - scoring weights and formula parameters from Gemini-aligned spec
2. REASON_CODES
   - full list of HF/SF codes + descriptions + typical retry step
3. RUNS
   - run_id, date, pipeline_id, character_id, move_id, total_frames, pass_count, reject_count, softfail_count, retry_rate, stop_reason
4. FRAMES

    - run_id, character_id, move_id, frame_index, attempt, seed/settings_hash, score, result, primary_reason_code, all_reason_codes, key metrics (ssim, dino, palette, baseline_error, lpips, etc.), file paths
5. CHARTS
    - pass rate over time
    - reject reason distribution
    - attempts-to-pass histogram
    - performance by pipeline_id (if present)
6. V0_V1_DIFF (optional)
    - show differences between original Gemini scoring and Opus-aligned scoring (weights, thresholds, gates)

B) Data Templates I can export and use immediately:

- frames.csv template (matching FRAMES tab columns)
- runs.csv template
- audit_result.json schema excerpt (aligned with Opus)
- "reason_code_enum.json" (a machine-readable list)

C) A "Python Ingestion Map"

- exact field names my Python agent should emit
- where each field lands in the workbook
- how to compute derived fields (retry_rate, stop conditions)

D) A "Threshold Calibration Plan"

- turn Opus's micro-tests into a 1–2 day calibration sprint:
  what data to collect, what charts to look at, how to adjust thresholds safely

RULES:

- Treat Opus spec as canonical for rubric/reason codes/schemas.
- Treat Gemini-aligned scoring as canonical for score math + gate behavior.
- Do not invent metrics—only include metrics present in the specs, plus clearly-labeled optional metrics.

Dashboard Workbook Created – 6 professional tabs ready for production sprite QA pipeline integration.

Tab Overview:

1. **CONFIG_THRESHOLDS** – Hard gates (HF-01 to HF-05), soft fail checks (SF-01 to SF-05), Gemini v2.0 scoring weights, rank bands, and tunable thresholds. This is your canonical source for all QA parameters.

2. **REASON_CODES** – Complete mapping of hard fail and soft fail reason codes with descriptions, typical retry steps, and knob adjustments. Use this to auto-route failures in your Python agent.
3. **RUNS** – Per-run summary: run_id, pipeline, character, move, frame counts, pass/reject/softfail rates, retry statistics, and stop conditions. Track overall pipeline health here.
4. **FRAMES** – Per-frame granular data: attempt counters, all computed metrics (ssim, dino, lpips, palette_delta, baseline_row, fringe_severity, etc.), result status, and reason codes. Feed this directly from your Python auditer.
5. **CHARTS** – Summary metrics, pass rate trends, and reject reason distribution. Auto-calculate from FRAMES/RUNS tabs to monitor pipeline drift.
6. **V0_V1_DIFF** – Side-by-side comparison showing how Gemini v1 (Opus-aligned) differs from v0 (original). Explains weight changes, formula updates, and retry ladder evolution.

Key Design Decisions:

- **Canonical Source:** CONFIG_THRESHOLDS is your single source of truth for all gates, thresholds, and weights
- **Reason Code Routing:** REASON_CODES + primary_reason_code in FRAMES enables deterministic retry ladder step selection
- **Metrics Completeness:** All metrics from Opus spec included (ssim, dino, lpips, palette, baseline, fringe, etc.)
- **No Invented Data:** Only metrics explicitly present in the Opus and Gemini specs are included

Now let me create the supporting data templates and Python ingestion map:

⁂

1. Opus-4.5-16bitfit-Audit-Rubric-Finalized.pdf
2. Gemini_Deep-Think_Sprite-Animation-Quality-Scoring-System-With-Opus-Pack.pdf
3. Gemini_Deep-Think_Sprite-Animation-Quality-Scoring-System.pdf