AI-Assisted Ticket Classification

High Level Design Document

Smart Triager Classifier (STC)

DC Ops | Oracle Cloud Infrastructure

February 2026

Author	Rik Kisnah
Team	DC Ops - Sudha's Team
Target Region	AGA (MI355X GPU Racks)
Version	1.0 - Draft
Classification	Internal

Table of Contents

1. Executive Summary

The AI-Assisted Ticket Classification system (STC - Smart Triager Classifier) is an LLM-bootstrapped pipeline that classifies DC Ops GPU failure tickets into structured categories. It combines deterministic rule-based matching with LLM reasoning and human-in-the-loop feedback to deliver accurate, repeatable categorization at scale.

The system enables TPMs and managers to categorize approximately 100 JIRA tickets in under 10 minutes, multiple times daily, replacing manual ticket-by-ticket triage with automated pattern recognition that surfaces operational hotspots for executive decision-making.

Key Distinction

STC is not RAG (Retrieval-Augmented Generation) or a pure LLM application. It is an LLM-assisted rule induction system where the AI bootstraps a deterministic rule engine that eventually operates without LLM involvement. The LLM is a training tool, not the production inference path.

2. Problem Statement

DC Ops teams managing GPU infrastructure (MI355X racks in the AGA region) handle a high volume of incident and service request tickets in JIRA. Without a systematic classification process, teams face several challenges:

Manual Triage Bottleneck: Engineers spend significant time reading and categorizing individual tickets, reducing time available for resolution.

Inconsistent Categorization: Different team members categorize similar failures differently, making trend analysis unreliable.

Delayed Hotspot Detection: Without automated categorization, recurring failure patterns take days or weeks to surface.

No Runbook Visibility: Tickets lack clear indicators of whether documented remediation procedures (runbooks) exist for the failure type.

Executive Reporting Gap: Managers lack real-time visibility into failure distribution for resource allocation decisions.

3. Solution Overview

STC addresses these challenges through a two-stage pipeline that progressively builds a deterministic rule engine from LLM-assisted pattern discovery and human expert validation.

3.1 System Architecture

The architecture follows a layered approach with clear separation between data ingestion, classification, human audit, and production deployment.

STC System Architecture			
STAGE 1: Data Pipeline		**STAGE 2: Classification & Audit**	
JIRA Ingestion REST API fetch with JQL filtering	**Normalization** Clean markup, split to per-ticket JSON	**Rule Engine + LLM** Hybrid classification with fallback	**Human Audit Loop** Review, correct, promote rules
Data Flow: JIRA API → Raw JSON → Normalized JSON → Rule Engine + LLM → Categorized CSV → Human Audit → Golden Rules → Executive Report			

4. Pipeline Stages

The system operates through two main stages containing nine discrete steps. Each step has defined inputs, outputs, and responsible actors.

Stage	Step	Name	Description	Output	Actor
1	1	**Fetch Tickets**	Extract tickets from JIRA REST API using JQL filters targeting GPU MI355X racks in AGA region. Supports date ranges, single ticket, or bulk pull.	tickets-json/	Automated
	2	**Normalize**	Clean JIRA wiki markup, remove bot comments and noise labels. Split paginated results into per-ticket JSON files optimized for LLM token efficiency.	normalized-tickets/<date>/	Automated
2	3	**Rule Engine**	Evaluate each	Matched rules +	Automated

		Eval	normalized ticket against the rule engine (regex pattern matching). The Rules Engine is the living knowledge base containing patterns from LLM proposals, human audits, and runbook indicators.	categories	
	4	**LLM Classification**	For tickets unmatched by rules, the LLM (Claude/OpenAI) analyzes ticket text and proposes a failure category with confidence score. Discovers reusable patterns for new rule proposals.	LLM categories + new rule proposals	LLM
	5	**Knowledge Base**	The accumulated knowledge base of logs from Runbooks, documented failure patterns, and Code-level indicators. Meta-rules detect runbook presence (TRS prescriptions, prescriptive actions, known bugs).	Runbook detection signals	System
	6	**Generate Output**	The LLM generates the categorized tickets CSV combining rule-engine matches and LLM classifications. Each ticket gets: failure category, confidence score, categorization source, and runbook status.	tickets-categorized.csv	LLM + Rules
	7	**Human Audit (Tickets)**	Human auditors review categorized tickets, marking each as correct, incorrect, or needs-review. Incorrect	Audited tickets CSV	Human

			categorizations include comments with the correct category.		
	8	**Human Audit (Rules)**	Human auditors review rule engine entries. Confirmed rules get confidence bumped to 1.0. Incorrect rules are narrowed, re-prioritized, or removed. New rules are proposed from feedback.	Audited rule-engine.csv	Human
	9	**Executive Report**	Categorized tickets are published with key findings: failure distribution, hotspot identification, runbook coverage gaps, and trend analysis for resource allocation decisions.	Executive dashboard / report	TPM / Manager

5. Core Components

5.1 The Rule Engine (Living Knowledge Base)

The Rule Engine is the central intelligence of STC. It is a CSV-based pattern store that grows over time through LLM discovery and human validation. Each rule contains a regex pattern, the ticket field to match against, the failure category to assign, a priority for evaluation ordering, and a confidence score reflecting its validation status.

Rules are evaluated in priority order (highest first). When multiple rules match a single ticket, all matching rule IDs are recorded, but the failure category is inherited from the highest-priority match. Confidence propagates as the maximum across all matching rules.

Rule Lifecycle

1. LLM proposes a new pattern (Confidence: 0.7, Created By: llm) → 2. Human reviews during audit → 3a. Confirmed: confidence bumped to 0.95-1.0, Created By: human-confirmed | 3b. Rejected: pattern narrowed, priority adjusted, or rule removed → 4. Promoted to Golden Rule Engine (read-only production copy)

Rule Engine Schema (rule-engine.csv)

Field	Description	Type	Notes
RuleID	Unique identifier	string	Sequential: R001, R002, R003...
Rule Pattern	Regex pattern	string	Case-insensitive matching. Supports alternation (|), lookahead, etc.
Match Field	Ticket field to inspect	string	summary, description, labels, comments, or combined (summary+description)
Failure Category	Specific failure label	string	Assigned when rule matches. Meta-rules use "Runbook Present = TRUE"
Category	High-level bucket	string	For reporting rollups: CDFP, GPU, Networking, Infrastructure, etc.
Priority	Evaluation order	int	Higher values evaluated first. More specific rules get higher priority.
Confidence	Validation score	float	1.0 = human-confirmed, 0.7 = LLM-suggested, < 0.5 triggers needs-review
Created By	Rule origin	string	human, llm, human-confirmed, human-feedback
Hit Count	Lifetime matches	int	Used to identify and prune unused rules

5.2 Classification Output

Each categorized ticket produces a row in tickets-categorized.csv with full traceability: the assigned category, which rules fired (if any), whether the classification came from a rule or LLM reasoning, the confidence level, and audit status fields for human review.

Field	Type	Description
Ticket	string	JIRA ticket key (e.g., DO-2639750)
Status	string	Current JIRA workflow status (Resolved, Open, In Progress)
Created / Age	date / int	Ticket creation date and days elapsed since creation
Runbook Present	bool	TRUE if meta-rules detect runbook indicators (TRS, prescriptive actions, known bugs)
Category of Issue	string	Failure category assigned to the ticket (from rule or LLM)
Category	string	Higher-level reporting bucket mirroring the rule engine
Rules Used	string	Comma-separated list of matching RuleIDs
Categorization Source	string	"rule" (deterministic match) or "llm" (AI reasoning) or "none" (uncategorized)
LLM Confidence	float	0.0–1.0 reliability score. Semantics depend on source (see Section 6)
Human Audit	string	Auditor verdict: correct, incorrect, needs-review, or pending-review
Human Comments	string	Auditor notes, corrections, or rationale

6. Confidence Model

Every categorization carries a confidence score (0.0–1.0) that drives automated triage. The interpretation depends on the categorization source:

Source	How Confidence Is Set	Implication
rule	Highest Confidence value among all matching rules. 1.0 = human-confirmed rule, 0.7 = LLM-suggested rule.	High confidence means the pattern has been validated by a human. The categorization is deterministic and reproducible.

llm	Model's self-reported confidence based on reasoning quality when no rule matched.	Lower reliability. The LLM may propose a new rule if it identifies a reusable pattern.
none	No confidence available. No rule matched and no LLM classification was performed.	Ticket is uncategorized. Requires human review or LLM-assisted analysis.

Automatic Triage Threshold

Tickets with LLM Confidence < 0.5 are automatically flagged as "needs-review" so auditors can prioritize low-signal items first. All other tickets default to "pending-review" until a human updates the verdict.

7. Human-in-the-Loop Design

The defining architectural principle of STC is that humans gate every rule promotion. The LLM is a powerful pattern discovery engine, but it cannot unilaterally change the production rule set. This design prevents rule drift from hallucinated patterns and ensures institutional knowledge is preserved.

7.1 The Feedback Loop

1 LLM Proposes	2 Human Reviews	3 Rules Updated	4 Regression Test	5 Golden Promotion	6 Production Use
LLM categorizes tickets and proposes new regex patterns	Human marks tickets as correct, incorrect, or needs-review	Confirmed rules get confidence boost; incorrect rules fixed or removed	Re-run categorization to verify rules reproduce audited results	Manual copy of stable rules to read-only golden directory	Rule engine categorizes all tickets without LLM involvement

7.2 Audit Verdicts and Actions

Verdict	When Used	System Action
correct	Categorization matches the expert's judgment	Bump rule confidence toward 1.0 (cap at 0.95 unless already higher). Set Created By = human-confirmed. Increment Hit Count.
incorrect	Wrong category, wrong runbook status, or false positive	Parse Human Comments for correct category. Widen narrow patterns, narrow broad patterns, adjust priority, or remove fundamentally wrong rules. Propose new rules if needed.
needs-review	Auditor uncertain, or auto-flagged by low confidence	Skip during rule training. Leave for deeper review. Ticket remains in queue.

8. Runbook Detection

A secondary objective of STC is identifying which tickets have documented remediation procedures (runbooks). This is implemented through meta-rules in the rule engine that only affect the Runbook Present field without influencing the failure category assignment.

Runbook Indicators:

Rule	Pattern	Signal
R011	"TRS prescription" or "Resolving for TRS" in comments	TRS-based automated remediation exists
R012	TRS_PRESCRIPTIVE_TICKET label	Ticket was auto-generated by TRS prescription system
R015	"Prescriptive Action" in ticket title	Title indicates a prescriptive (runbook-driven) action
R017	"RHS Hardware Update/Action Plan" in comments	RHS system has documented action plan
—	"known bug" in ticket text	Issue is documented in CHS (known bug database)

9. Technology Stack

Component	Technology	Purpose
Data Source	JIRA REST API	Ticket extraction with JQL-based filtering
Ingestion Scripts	Python 3 (requests, argparse)	Fetch, normalize, and compact ticket data
Rule Engine	CSV + Python regex (re module)	Deterministic pattern matching and categorization
LLM (Training)	Claude (Anthropic) / OpenAI	Pattern discovery, fallback classification, rule proposals
LLM Interface	Claude Code CLI	Session-based training with prompt files
Storage	File-based (JSON + CSV)	Normalized tickets and rule/category data
Version Control	Git with snapshot archival	Pass-suffixed CSVs preserve audit trail
Package Manager	uv (Astral)	Python dependency management

10. Directory Structure

Path	Access	Purpose

scripts/tickets-json/	Read/Write	Raw JIRA JSON (Step 1 output)
scripts/normalized-tickets/<date>/	Read/Write	Normalized per-ticket JSON (Step 2 output)
scripts/trained-data/	Read/Write	Working directory for training sessions (Steps 3–4)
scripts/trained-data/golden-rules-engine/	**READ-ONLY**	Production rule engine (Step 5). Never modified by automation.
scripts/analysis/	Read/Write	Output from bulk categorization runs (Step 6)
templates/	**READ-ONLY**	CSV header-only skeletons. Never write working data here.
prompts/	Read	LLM prompt files for each workflow stage

11. Design Principles

1. Human-Gated Rule Promotion: The LLM proposes patterns, but humans approve before any rule enters the production rule engine. This prevents rule drift from hallucinated matches and ensures domain expertise is preserved.

2. Confidence-Driven Triage: Scores below 0.5 automatically flag tickets as needs-review. Auditors focus on low-signal items first, maximizing the value of limited human review time.

3. Dual-Source Tracking: Every categorization records whether it originated from a deterministic rule or LLM reasoning, with separate confidence semantics for each. This enables measurement of rule engine coverage over time.

4. Meta-Rule Separation: Runbook detection rules only set the Runbook Present field and never influence failure category assignment. This prevents runbook signals from contaminating the classification taxonomy.

5. Snapshot Archival: Pass-suffixed CSV files preserve the complete audit trail across training iterations. Any past state can be restored and compared for regression testing.

6. Deterministic Reproducibility: Tickets are processed in filename order and rules are evaluated by priority. Given the same inputs and rule engine, the output is identical every time — enabling regression verification.

7. Progressive Autonomy: The system is designed to reduce LLM dependence over time. As the rule engine matures through human feedback, more tickets are classified deterministically without LLM involvement, reducing cost and latency.

12. Success Metrics

Metric	Target	Measurement
Categorization Time	< 10 minutes for 100 tickets	End-to-end time from ticket fetch to categorized output
Rule Engine Coverage	> 80% of tickets matched by rules	Percentage of tickets with Categorization Source = rule
Classification Accuracy	> 90% correct on human audit	Percentage of audited tickets marked as correct
Runbook Detection Rate	100% of known runbook tickets identified	False negative rate on Runbook Present field
Rule Stability	Zero regression failures	Audit regression pass rate before golden promotion

13. How to Use STC — Operational Guide

This section provides step-by-step instructions for running the STC pipeline. The process is designed for daily use by TPMs and operators who need to categorize incoming tickets and refine the rule engine over time.

13.1 Prerequisites

STC requires the uv package manager (https://docs.astral.sh/uv/) and Python 3. Initial setup:

Command	Purpose
uv init	Initialize the project environment
uv add requests	Install HTTP library for JIRA API calls
uv add --dev pytest	Install test framework (development only)

13.2 Daily Workflow (5 Steps)

The standard operating procedure follows five steps. Steps 1–3 are run sequentially. Steps 4–5 form an iterative feedback loop that is repeated until the rule engine stabilizes.

Step	Action	Details
1	**Fetch Tickets**	Pull raw JSON from JIRA. Common usage patterns: `uv run python scripts/get-tickets.py` — Fetch all tickets `uv run python scripts/get-tickets.py -2d` — Last 2 days `uv run python scripts/get-tickets.py 2025-01-01 2025-01-31` — Date range `uv run python scripts/get-tickets.py -t DO-2639750` — Single ticket
2	**Normalize Tickets**	Transform raw JIRA JSON into compact per-ticket schema: `uv run python scripts/normalize-tickets.py`
3	**Run Rule Engine**	Categorize all normalized tickets using the golden rule engine: `uv run python scripts/rule-engine-categorize.py` Output: scripts/analysis/tickets-categorized.csv
4	**Audit Results**	Review tickets-categorized.csv. Look for: • Rows with LLM Confidence < 0.5 or Human Audit = needs-review • Tickets marked "uncategorized" or with clearly wrong categories • Cases where a rule should have matched but RuleID is missing • Repeated patterns across tickets that should become new rules

		Then update rules via LLM-assisted feedback: `./scripts/run-update-rules.sh`
5	**Verify & Iterate**	Re-run Step 3 to verify the updated rules produce expected results: `uv run python scripts/rule-engine-categorize.py` If results are satisfactory, promote to golden. Otherwise repeat Steps 4–5.

13.3 Script & Prompt Reference

Task	Script	Prompt
Fetch tickets from JIRA	**scripts/get-tickets.py**	n/a
Normalize tickets	**scripts/normalize-tickets.py**	n/a
Categorize (rules only)	**scripts/rule-engine-categorize.py**	n/a
Train / categorize (LLM batch)	**scripts/run-training.sh**	prompts/train-to-categorize-tickets-prompt.md
Update rules from feedback	**scripts/run-update-rules.sh**	prompts/update-rule-engine-prompt.md

13.4 Glossary: Audit Status Values

The Human Audit for Accuracy field tracks the review lifecycle of each categorized ticket. It is set automatically by the pipeline and updated manually by a human auditor.

Value	Set By	Meaning
pending-review	Pipeline	A rule matched with confidence ≥ 0.5. Categorization is likely correct but has not been verified by a human yet.
needs-review	Pipeline	No rules matched (uncategorized) or confidence was below 0.5. These tickets need priority human attention because the system is uncertain.
correct	Human	Auditor confirmed the categorization is accurate. Rule confidence will be bumped on next training pass.
incorrect	Human	Auditor determined the categorization is wrong. Details should be provided in Human Comments for rule correction.

14. Future Considerations

Dashboard UI: A React-based dashboard with Recharts for real-time visualization of failure distribution, trend analysis, and rule engine coverage metrics.

Batch API Integration: Leverage Claude's Batch API for cost-effective bulk classification of large ticket backlogs during off-peak hours.

Database Migration: Move from file-based CSV storage to SQLite or PostgreSQL for better concurrent access, querying, and aggregation capabilities.

Automated Scheduling: Implement cron-based scheduling for periodic ticket ingestion and classification (currently supported via run-overnight.sh).

Multi-Region Expansion: Extend the JQL filters and rule engine to cover additional Oracle regions beyond AGA.

Rule Engine Versioning: Formal version tagging of golden rule engine releases with changelog and rollback capabilities.