# Hybrid NLP + LLM Architecture for Network Automation (User -> Cisco CLI -> User) with Active Learning Loop

This document describes a complete, production-ready architecture for converting user natural language to Cisco switch CLI commands and turning CLI output into user-friendly text. It includes a hybrid NLU design (lightweight models + deterministic templates + LLM fallback), system components, data flows, API definitions, folder structure, deployment notes, monitoring, testing guidance, and now **an active learning loop** to continuously improve from user feedback.

# Overview

**Goals:** - Convert user instructions (NL) into safe, accurate Cisco CLI commands. - Parse Cisco CLI outputs and present concise, human-friendly responses. - Minimize LLM usage for cost/latency/consistency; use it as a fallback or for “prettifying” outputs. - Provide deterministic, auditable command generation for safety. - **Continuously learn from user corrections and confirmations to improve intent/entity extraction and command mapping over time.**

**High-level approach:** hybrid NLU + active learning: 1. Lightweight Intent Classifier 2. Named Entity Recognition (NER) 3. Rule-based command generator 4. TextFSM parser for CLI outputs 5. Optional LLM for natural summaries 6. **Feedback ingestion pipeline to retrain models periodically with user-confirmed mappings and corrections.**

# Active Learning Flow

1. **User Interaction:** User gives an instruction or reviews a generated command/output.
2. **Feedback Capture:** The system logs corrections, confirmations, or edits.
3. **Data Labeling:** Feedback is tagged automatically (confirmed = positive example, corrected = paired positive/negative examples).
4. **Training Data Store:** Stores all labeled examples in a version-controlled dataset.
5. **Periodic Retraining:**
   * Intent classifier and NER models retrained on updated dataset.
   * Rule templates adjusted based on common corrections.
6. **Deployment:** New models deployed after evaluation against a validation set.

# Components

* **NLU Service:** Handles intent classification + NER.
* **Command Generator:** Maps intent/entities to CLI templates.
* **Parser Service:** Converts CLI output to structured data.
* **Summary Service:** Converts structured data to human text (LLM optional).
* **Feedback API:** Receives user feedback on commands and summaries.
* **Model Trainer:** Batch retrains models using feedback data.
* **Data Store:** Maintains labeled examples, raw logs, and model versions.

# Benefits of Active Learning

* Improved accuracy over time without full manual dataset labeling.
* Adaptation to new command types and syntax.
* Continuous alignment with real-world usage patterns.

# Active Learning Loop (Addendum)

This addendum upgrades the architecture into a **continuous, human-in-the-loop active learning system**. It captures new user inputs, operator corrections, and device execution outcomes to **auto-improve** the intent classifier, NER, parsing templates, and safety rules over time—safely and with auditability.

## New/Updated Components

* **Feedback Ingestion & Label Store**
  + Collects supervision signals:
    - User clarifications (missing entities / ambiguous intents)
    - Operator edits to commands before execution
    - Execution outcomes (success/failure, device errors)
    - Parser failures and manual fixes (edited TextFSM outputs)
    - Post-run thumbs-up/down and free-text notes
  + Stores raw samples, derived labels, and metadata in **Postgres** (labels) + **object storage** (raw text/outputs).
* **Labeling UI (Ops Console)**
  + Queues low-confidence or failed cases for annotation.
  + Pre-populates suggestions (weak labels) from current models.
  + One-click approve/edit for intents, entities, and parsed JSON.
* **Active Learning Orchestrator**
  + Sampling strategies to pick informative examples:
    - **Uncertainty sampling:** low margin / entropy from intent & NER
    - **Disagreement sampling:** small **committee** of checkpoints/discrete models
    - **Error-driven:** parser/exec failures and operator-edited commands
    - **Diversity sampler:** avoid near-duplicate paraphrases
  + Batches examples into **training sets** and triggers scheduled retrains.
* **Training Pipeline & Model Registry**
  + Automated pipelines (e.g., on Airflow/GitHub Actions) for:
    - Data validation → feature generation → fine-tune → evaluate → package
  + **Model Registry** (e.g., MLflow): versioned models with metrics, datasheets, licenses.
  + **Canary/Shadow** deploys via the NLU Service for safe rollouts.
* **Policy Auto-Guardrails Updater (Optional)**
  + Proposes updates to blocklists/allowlists based on near-missed incidents or operator overrides (requires human approval).

## Updated Data Flow (with Feedback)

1. **Confidence Gate**: NLU returns scores; if < threshold or conflicting intents, route to LLM clarify OR human review.
2. **Operator Edit Capture**: any manual edit to CLI commands is diffed and stored as supervision for next training round.
3. **Execution Telemetry**: device-side errors (e.g., invalid interface) are logged and linked to originating sample.
4. **Parser Feedback**: parser failures + successful manual regex/TextFSM fixes become new templates or test cases.
5. **User Feedback**: thumbs-up/down and notes are tied to the sample for quality signals.
6. **Batch & Retrain**: Active Learning Orchestrator periodically selects samples and kicks off retraining.
7. **Safe Deployment**: new models run in **shadow** (score-only) and **canary** modes with rollback on regressions.

## APIs (Additions)

* **POST /api/v1/feedback**  
  Body: { sample\_id, stage: "nlu|template|exec|parser|summary", signal: "good|bad", notes?, suggested\_labels? }
* **POST /api/v1/labels**  
  Body: { sample\_id, intent, entities, reviewer\_id } (from Labeling UI)
* **POST /api/v1/training-jobs** (admin)  
  Body: { dataset\_version, task: "intent|ner|parser", strategy: "uncertainty|disagreement|error" }
* **GET /api/v1/models** / **POST /api/v1/deploy**  
  Manage model versions and rollout strategies (shadow/canary/all).

## Active Learning Strategies (Details)

* **Uncertainty Thresholds**: e.g., intent margin < 0.15 or NER mean token prob < 0.85 → send to queue.
* **Committee Disagreement**: maintain 3 light checkpoints; if predicted intents differ → annotate.
* **Execution-Aware Labeling**: failed device commands that were auto-corrected by operators are high-priority positive labels.
* **Template Drift Detection**: monitor parser failure rates per command; auto-suggest new/updated TextFSM templates.
* **Replay Buffer**: hold out a stable validation set to prevent catastrophic forgetting.

## CI/CD & Safety Gating

* **Pre-deploy checks**: must exceed baseline on F1 (NER), accuracy (intent), and **end-to-end command correctness**.
* **Shadow testing**: compare predictions vs. production model on live traffic without execution side-effects.
* **Canary**: 5–10% traffic; auto-rollback if error budget exceeded (e.g., +1% parse failures or +0.5% operator edits).
* **Red Team**: periodic adversarial prompts to test safety policies and blocklists.

## Monitoring (Additions)

* Label throughput & age (time-to-annotation)
* % traffic below confidence threshold
* Parser failure rate & template update velocity
* Operator edit rate (proxy for model drift)
* Model version attribution on every decision (for auditing)

## Data & Privacy

* PII scrubbing on logs and raw samples; configurable retention.
* Role-based access to labeling UI and raw device outputs.
* License tracking for any third-party data or templates.

## Quick Start (Active Learning)

1. Add Feedback Ingestion table(s) and wire up **/feedback** from frontend + backend.
2. Implement basic uncertainty sampling and a daily retrain on the last N informative samples.
3. Stand up a minimal Labeling UI page in the Ops Console.
4. Enable shadow deployment of new NLU checkpoints before canary.
5. Add unit tests that replay prior failures; block deploy if they regress.

Note: The original doc already mentioned “Continuous learning”. This addendum turns it into a **concrete, automated active-learning loop** with safety gates and human oversight.