ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

Cisco Packet Tracer Troubleshooting Assistant

Ameek Sharma[1]; Anirudh Sharma[2]; Muskan Gyanani[3]; Bersha Kumari[4]; Madhu Choudhary[5]

[4,5]Assistant Professor

[1,2,3]Computer Scinece and engineering, Poornims Institute of Engineering and Technology Jaipur,India
[4,5]Department of Computer Scinece and engineering, Poornims Institute of Engineering and Technology Jaipur,India

Publication Date: 2026/04/09

Abstract: New complexity introduced to enterprise networks drives demand for advanced diagnostic tools that can not only quickly parse device configurations and detect misconfigurations, but also recommend remedies. Network troubleshooting in traditional environments is still a manual, time-consuming process that relies heavily on expert know-how. In this paper, we present a new architecture for an Intelligent Network Troubleshooting Assistant that automates the entire end-to-end diagnostic pipeline. It uses a multi-stage parsing engine that can parse heterogeneous Cisco IOS outputs, a deterministic rule engine to match against eight classes of misconfigurations, produces a weighted health score and associated fix plan. The system architecture was inspired by the Model Context Protocol (MCP) and microservices architecture, where its modularization, scalability, and educational purpose among other design principles were taken into consideration. A multi-device, complex-network, case study validates our system's ability to detect root-cause issues accurately and produce educational insights and actionable remediation steps. Our findings indicate that the architecture not only lowers mean-time-to-resolution (MTTR) for network problems but also acts as a tool to teach budding network engineers.

Keywords: *Network Troubleshooting, Cisco IOS, Rule-Based Systems, Health Scoring, Educational Technology, Microservices, Model Context Protocol.*

How to Cite: Ameek Sharma; Anirudh Sharma; Muskan Gyanani; Bersha Kumari; Madhu Choudhary (2026) Cisco Packet Tracer Troubleshooting Assistant. *International Journal of Innovative Science and Research Technology*, 11(4), 94-102. https://doi.org/10.38124/ijisrt/26apr198

I. INTRODUCTION

Network engineering is undergoing a revolution. As enterprise networks become greater in scale, and higher in complexity, the need for fast, precision automated troubleshooting is more important than ever! Command-line interface (CLI) outputs are extensive and common with routers, switches, firewalls, etc., where network administrators have to manually parse the text, correlate the information together and make assumptions on root cause. This is a long process and prone to human error, particularly for students or junior engineers [1].

Conventional network management systems typically leverage SNMP traps, or centrallized logging, both of which are prone to high friction on capturing the more subtle configuration slippages that only show in certain CLI outputs. Furthermore, these systems often show the data in a compartmentalized way with correlation and diagnostics left up to human minds. The challenge is compounded in education environments where students need to learn directions to interpret CLI outputs without instant access to expert guidance [2].

This paper presents the design and implementation of a Network Troubleshooting Assistant, a full-stack web application that automates the analysis of raw Cisco IOS CLI outputs. The system is built upon a multi-stage pipeline comprising a parser, rule engine, health scoring module, and an analysis assistant. The goal is to transform raw, unstructured CLI text into structured insights, prioritized action plans, and educational content.

The architecture of this system aligns with emerging paradigms in AI agent design, specifically the Model Context Protocol (MCP), which standardizes how AI systems interact with external tools and data sources [3]. By treating parsing modules, diagnostic rules, and scoring algorithms as independent, composable components, the system achieves a level of modularity and maintainability that is often lacking in monolithic diagnostic tools.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

The key contributions of this paper are:

- *A Robust Multi-Device Parser:*
  A flexible parsing engine capable of segmenting mixed CLI outputs from multiple devices and extracting structured data from five key Cisco IOS commands.

- *A Comprehensive Rule Engine:*
  A set of eight deterministic diagnostic rules that identify common misconfigurations across interfaces, routing, VLANs, and IP addressing.

- *A Weighted Health Scoring Model:*
  A quantitative metric (0–100) that provides an immediate assessment of network health, broken down by functional categories.

- *A Prioritized Fix Plan Generator:*
  A mechanism that ranks identified issues by root-cause priority, providing specific Cisco IOS commands for remediation.

- *A Pedagogical Layer:*
  Educational insights designed to help users understand the underlying networking concepts and the rationale behind each fix.

By integrating these components, the proposed system addresses the critical need for intelligent, automated, and educational network diagnostic tools.

II. LITERATURE REVIEW

- *Evolution of Network Management*
  Traditionally, network management consists of five functional areas: fault, configuration, accounting, performance and security management (FCAPS) [4]. They used SNMP-based approach for monitoring initially, which enables standard data collection across network devices. But SNMP doesn't even come close when it comes to expressing the state of a device as viewed through commands in the CLI like show ip route or show running-config.

CLI based trouble shooting is still very much alive though and for good reason, it has depth and is universal. “The engineers have to ssh into devices, run the commands and interpret the output. While effective, this process is inherently reactive and non-automated - not to scale in large, multi-vendor environments [5].

- *Automation in Network Diagnostic*
  Recent advances introduced automation in network management. To provide programmatic access to network devices we have tools such as Ansible and Python libraries (Netmiko, NAPALM) While these tools can push configurations and gather data, they often require considerable scripting to incorporate diagnostic logic[6].

Network anomaly detection has also been one of the features explored using machine learning (ML). Unusual patterns in traffic or behaviors of devices can be detected using techniques like clustering and classification. However, machine learning (ML) models tend to be data-hungry with their training often requiring large tagged datasets and can often produce “black box” outputs that are not easily interpretable, a key limitation for troubleshooting techniques in environments where understanding the why is necessary as much as the how [7].

- *Rule-Based Diagnostic Systems*
  Rule-based systems provide a deterministic, humanly-interpretable alternative for validating configurations. Such systems are embedded with expert knowledge in the form of "if-then" rules operating on configured data, based on their structure. They are best used in scenarios where there are clear failure modes, such as ensuring certain interfaces are up or down, verifying that a particular route was learned and is present in the routing table or making sure the VLANs we expect are consistent [8].

The solution also proposed a system that builds upon this idea using rule-based diagnostics, a health score and prioritized fix plan. It solves the issues of both manual troubleshooting (slow and prone to errors) and ML-based approaches (dark box and data-hungry).

- *The Evolution of Agentic Artificial Intelligence*
  Artificial Intelligence is transitioning away from reactive generative models towards autonomous programs called AI agents that are oriented at optimizing for specific goals. Unlike normal LLMs that write text per its token of training data, agents can now reason, plan and execute multi-step tasks in a relatively autonomous manner.

An agentic system is one able to reach out into the outside world via “tools” — running scripts, querying databases etc. Without this external ability an agent is merely a cognitive shell with its training set the last best configuration it can ever learn. "For the networking assistant, this means that the agent is expected not only to 'understand' networking theory but also be capable of 'seeing' and 'interacting with' in real time with Cisco Packet tracer environment."

- *Security and Privacy in Network Tools*
  Security and privacy have become paramount with the advent of automated tools. Network diagnostic tools deal with sensitive configuration information, like passwords, IP addresses and network topology details. One other critical consideration is the adoption of zero-trust security principles in which no component should ever be blindly trusted [9]. To mitigate this concern, our system enforces user authentication and role-based access control to restrict access to sensitive data, ensuring that only authorized users can access or export analysis results.

Also, the maintainability of these server is an area studied. As this ecosystem grows, mainstream adoption of MCP servers has shown the necessity for a certain degree of standardized governance to protect against quality control and security threats. Our proposed architecture maintains a distinction between the base diagnostic toolsets and the remainder of the agentic framework in order to maintain such

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

isolation for modular, secure design pares per microservices tenants, ensuring that updates are localised, not global.

III. METHODOLOGY

To address the challenges of manual troubleshooting and existing tools, we propose NTA (Network Troubleshooting Assistant), a modular architecture based on microservices. Taking raw CLI text as input, it extracts structured data out of it and uses diagnostic logic to form some actionable insights.

➢ *Fundamental Architectural Principles*

The NTA is built on four key principles:

- *Modularity:*

You have the major functions; parsing, rule evaluation, scoring and insight generation are independent modules It also allows to develop, test and upgrade independently.

- *Deterministic:*

Diagnostic method is rule based and deterministic. Same input always generates the same output, making it reliable and explainable. From unstructured CLI text to structured data (JSON) that can be stored, queried and analyzed.

- *Educational Value:*

Not only diagnostic, the system offers explanations and learning material for every detected issue.

➢ *System Components*

The NTA architecture consists of five core components:

- *Parser module:*

This is the first component to analyze the raw user input. It tries to intelligently break the output from multiple devices into segments by looking for hostnames, prompts, or separators. It calls five specialized parsers for each device to parse show ip interface brief, show ip route, show vlan brief, and the router sections of both show running-config (interface) and show running-config (router).

- *Rule Engine:*

This stage contains eight diagnostic rules (explained in Section III-C) that are iterating through the extracted data. Each rule returns a structured issue object consisting of device, interface, severity, description and recommended command fix.

- *Health Scoring Module:*

This process gives a numerical health score ranging from 0 to 100. It begins with a perfect score and deducts points for every issue that is discovered, depending on severity (critical, high, medium, low). Individual deductions are sized in proportion to their importance by category (Routing: 30%, Interface: 25%, VLAN: 25%, IP: 20%) relative to total loss of function.

- *Analysis Assistant:*

This component generates three key outputs:

✓ *Evidence Coverage Report*

Analyzes how complete the user input was, looking for presence of four essential commands and suggesting those that may be missing.

✓ *Prioritized Fix Plan:*

Organizes issues by type of failure and assigns a priority weight based on the root-cause. This correctly gets into the fact that issues which are underlying (e.g. a physical link is down) should be handled before an apparent issue (missing routes).

✓ *Educational Insights:*

Provides deterministic, human-readable explanations, such as the order in which to fix issues and the purpose of specific fix commands.

- *Storage & API Layer:*

The SQLite database holds all of the analyses, any raw input, parsed data, issues encountered and outputs generated. This data is exposed to the frontend (a React-based Single Page Application) through a RESTful API (built using Flask).

IV. DISCUSSION

The Network Troubleshooting Assistant represents a significant step forward in automating network diagnostics. Its modular, rule-based approach addresses the core weaknesses of both manual and ML-driven methods. The system's architecture provides several key benefits:

➢ *Enhanced Efficiency:*

By automating the parsing and correlation of CLI outputs, the system reduces the time required for diagnosis from minutes or hours to seconds. This is particularly valuable in time-sensitive environments or during network outages.

➢ *Reduced Cognitive Load:*

The system displays information in an organized way that helps prioritize alerts. Instead of having to sift through hundreds of lines of text, a user can immediately view a list of prioritized issues along with the devices and interfaces that are affected and precisely what commands need to be executed in order to fix them.

➢ *Reliable & Consistent Diagnosis:*

Making use of the rule-based approach, we can ensure diagnosis that is reproducible and do not have errors associated with humans. When the same input always yields the same output, it serves as a constant baseline for analysis.

But the system has its shortcomings. Its dependency on deterministic rules is its biggest weakness. Though this makes the system interpretable, it also means that it will be unable to recognize new or complex problems outside of the written ruleset. It cannot diagnose a routing loop, for example, introduced via misconfigurations of dynamic routing protocols not directly represented in the show ip route output.

Its reliance on complete user input is another limitation. The coverage report for evidence helps alleviate this

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

somewhat by inferring commands that may have been issued, but how good the analysis will be performed is directly related to how well the CLI command outputs are provided.

Next focus area will be growing the set of rules to incorporate advanced topics such as OSPF/EIGRP peer relations, ACL verification and NAT configs. Implementing LLM for free-form analysis, by leveraging the Model Context Protocol (MCP) to have the system deal with ambiguous and resort to incompletely specified inputs more smoothly and give greater depth of explanation [3] could be beneficial.

V. IMPLEMENTATION: CISCO PACKET TRACER MCP DIAGNOSTIC SERVER

The proposed solution follows a client-server architecture where the MCP Server acts as the backend diagnostics engine.

➢ *Integration and Monitoring Modules*

Integration Module: Gathers the network topology, device_configs and packet simulation results from Cisco Packet tracer environment. It then processes this data to create a network model that makes logical sense. The server begins simulating the flow of packets (for example an ICMP echo request) through the network and step by step, follows the path which those packets take.

➢ *Diagnostic Engine and Solution Generation*

If a packet gets dropped the Diagnostic Engine looks at the state of the device when it was failing and compares routing tables, VLANs, port status etc. with properties of that packet. Solution Generation Module: This modules creates the diagnosis in plain English and one or many possible solutions listed with commands, using CLI for the problem diagnosed.

➢ *Technical Tools*

- *Cisco Packet tracer:*
  Core emulation environment.

- *Python/Node.js:*
  server-side development due to strong networking libraries.

- *Express.js:*
  Utilized to create an API for user interaction.

The project was a website where there are several options:

✓ The home page comes with several options to diagnos the problem facing in cisco packet tracer.

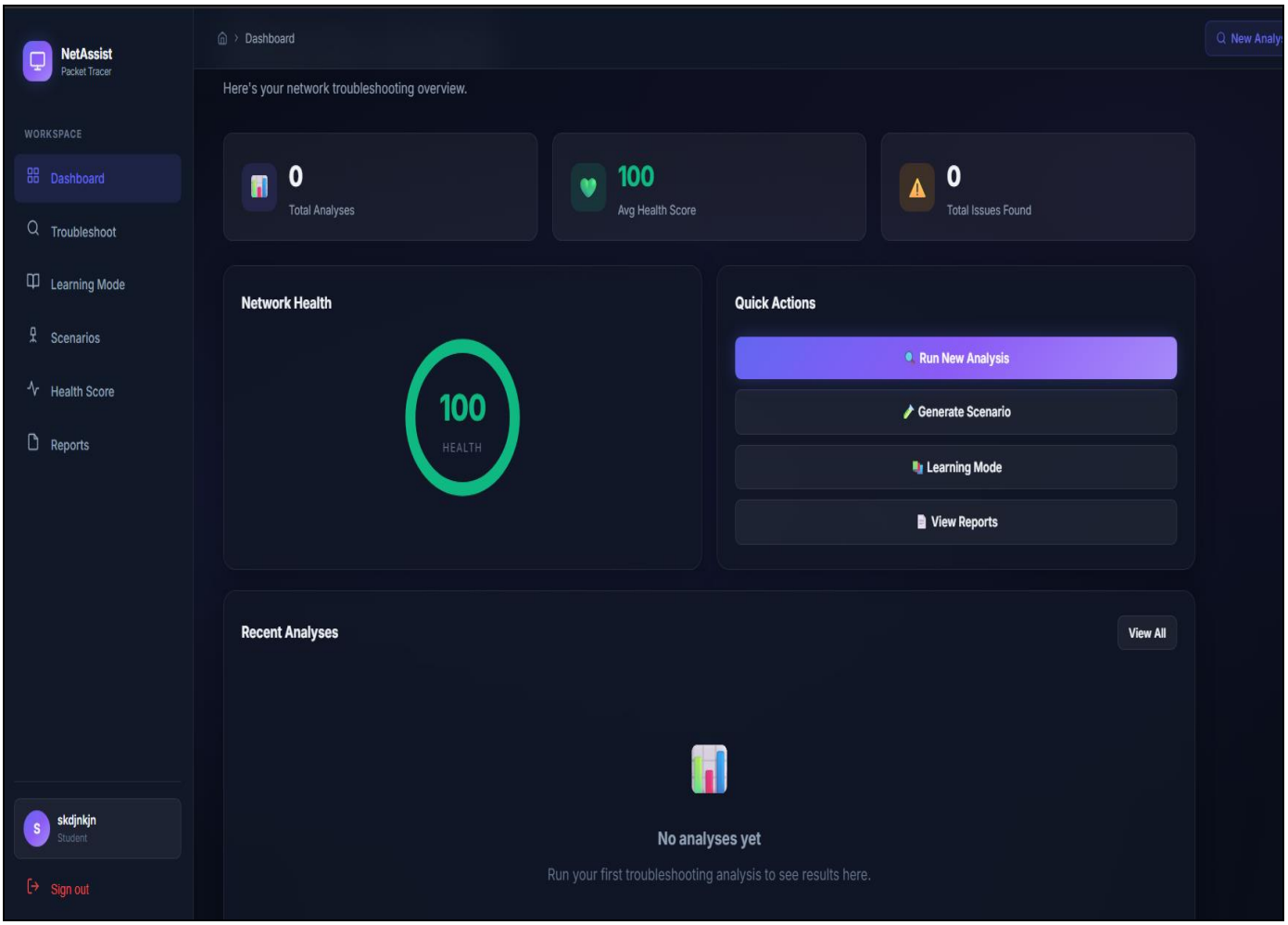

Fig 1 Home Page

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

✓ *Scenario Generation*:
we can genereate the scenarios for our understanding in which this project works.

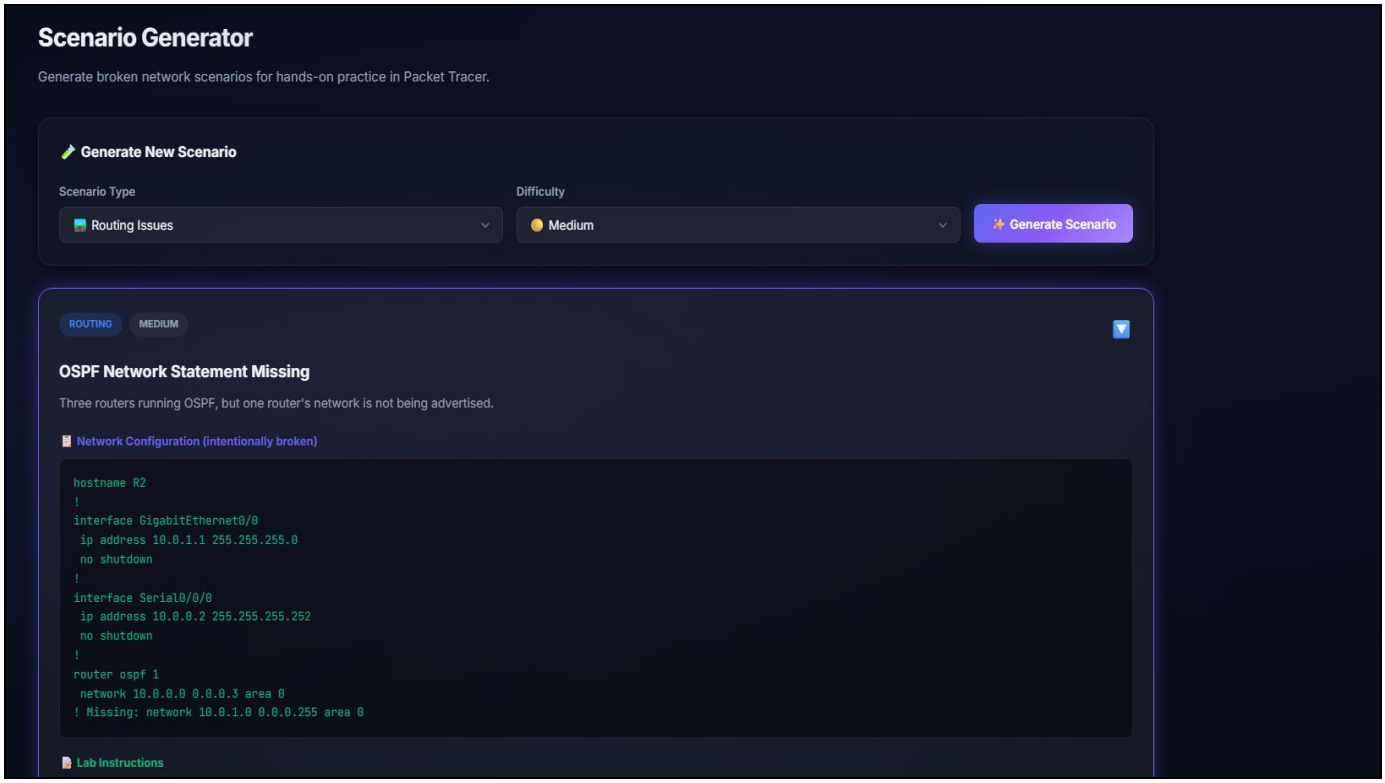

Fig 2 Scenario Generation

✓ After this we have the diagonstic option where we can solve our problems regarding Cisco Packet tracer.

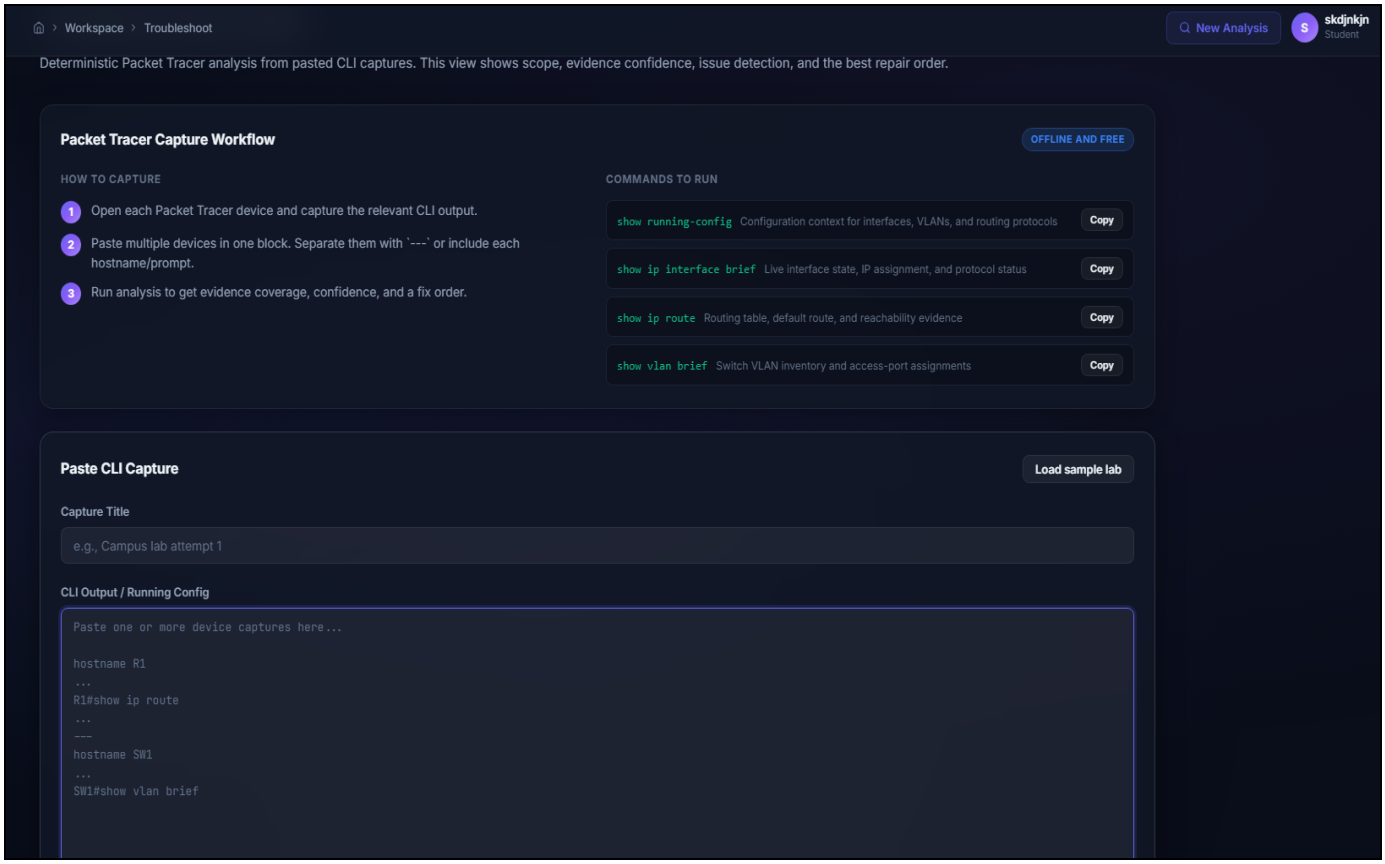

Fig 3 Diagnostic screen

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

✓ Lastly the health check where our solution occurs and we can check how optimized our design is.

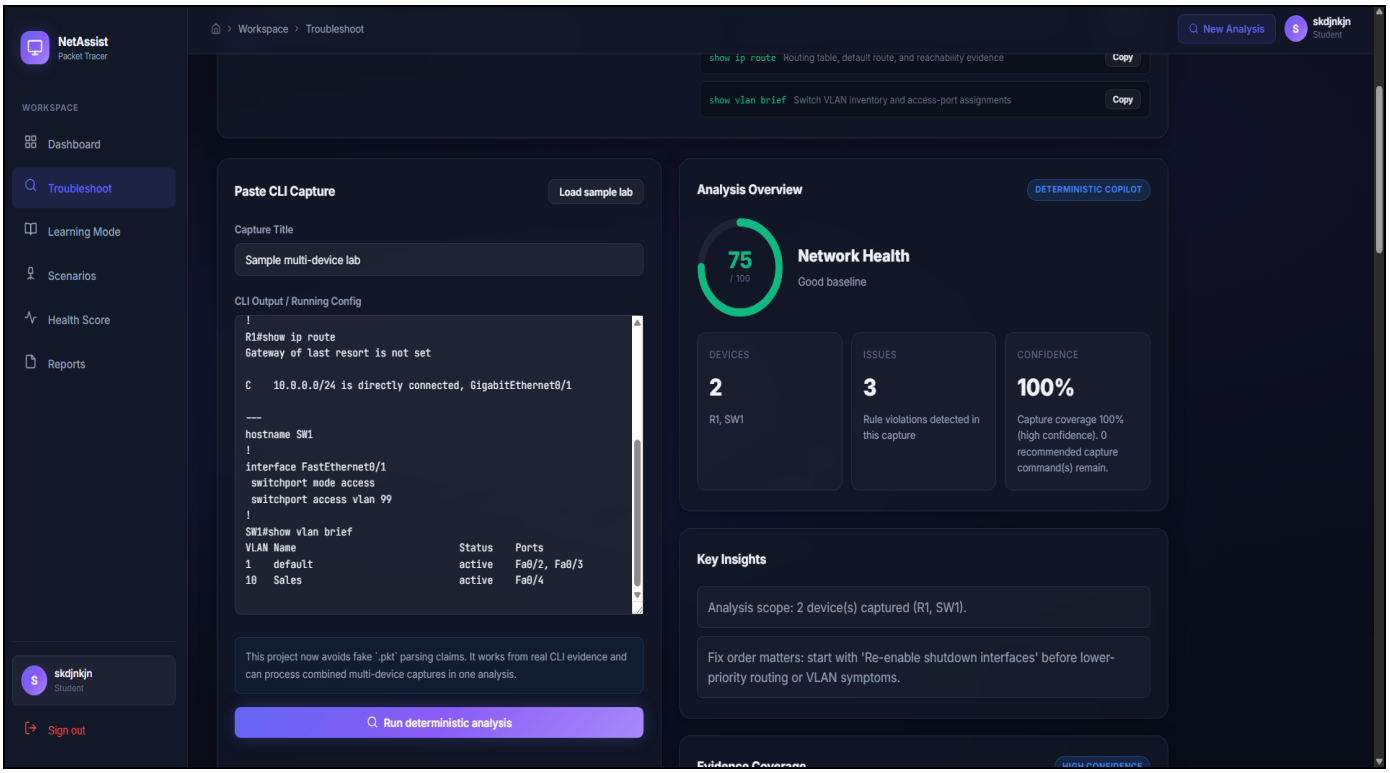

Fig 4 Health Check Page

PROJECT WEBSITE: https://netassist-debug-fyp.vercel.app/

VI. COMPARATIVE ANALYSIS: BEFORE AND AFTER THE PROPOSED SOLUTION

To have a full appreciation of the contribution of the Network Troubleshooting Assistant, the state of network troubleshooting prior to this solution and the revolutionary changes it brings need to be understood.

➢ *Before the Proposed Solution: The Traditional Approach*

Before the NTA, network troubleshooting, especially in educational and small-to-medium enterprise settings was a manual and labor-intensive process. This section describes the before state in five major dimensions.

- *Manual Parsing and Correlation:*

A network problem being faced by an engineer or student would normally log into each individual device (through console, SSH, or Telnet) and would run a series of CLI commands (e.g., show ip interface brief, show ip route, show running-config) and would then invariably scan the output to identify problems.

This was done by the person having to:

✓ Identify the normal vs. abnormal output pattern in memory.
✓ Compare data between several commands and devices.
✓ Keep records of findings, mental or written.

This process might take 30-60 minutes easily in case of a network with only three routers and two switches. In bigger networks, time was multiplicative.

- *Barrier of the Expertise:*

Quality troubleshooting demanded expertise based on experience. A junior engineer or student did not always have the mental models to: Learn to differentiate between root causes and symptoms. The sequence of which commands to use. Know the correct fix commands to use in a particular failure mode. This knowledge barrier formed a high learning curve and caused frustration especially in academic contexts where the learner was supposed to learn via errors without receiving expert advice at the first instance.

- *Unreliable and Inaccurate Diagnoses:*

Diagnosis was based on human vigilance, so it was inconsistent in nature. The same CLI output may be analyzed by two different engineers and lead to different conclusions. Common errors included: Missing an important problem that was lost in hundreds of lines of output. Failure to perceive the condition of an interface or route. Correcting a symptom (e.g. adding a static route) without correcting the underlying cause (e.g. a broken physical link).

- *No Quantitative Health Assessment:*

There was no objective, repeatable measure of network health presented by traditional methods. Engineers used qualitative evaluations (the network appears stable), or binary evaluations (it is working/ not working). This lack of quantification made it difficult to: Follow-up progress.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

Compare the degree of various problems. Support remediation to management.

- *Low Pedagogical Importance:*

During the process of troubleshooting, although it was identified to be a valuable learning activity, the conventional process did not provide much structured guidance. Those students who were unable to diagnose some issue had no means to know why they did not pass or what they overlooked. The learning opportunity was lost frequently. Table III is a summary of the weakness of the traditional approach.

- Below Table summarizes the limitations of the traditional approach.

➢ *Limitations of Traditional Network Troubleshooting*

Table 1 Traditional Network Troubleshooting Limitation

Dimension	Limitation
Time Efficiency	Manual parsing of CLI outputs; 30–60+ minutes per incident
Expertise Requirement	Deep, experience-based knowledge required
Consistency	High variability between different analysts
Quantification	No objective health metric
Pedagogy	No structured feedback or guidance
Scalability	Linear degradation with network size

➢ *After the Proposed Solution: The Transformed Approach*

The Network Troubleshooting Assistant fundamentaly transforms each of these dimensions.

Table 2 Before vs. After Comparison-1

Dimension	Before (Traditional)	After (NTA)
Time to Diagnosis	30–60+ minutes	Seconds
Expertise Required	High (expert-level)	Low (any user)
Consistency	Variable,human-dependent	100% deterministic
Health Quantification	None (qualitative only)	0–100 weighted score
Pedagogical Support	None (trial and error)	Explanations + fix commands
Scalability	Linear degradation	Constant (independent of network size)

- *Reduction in the Dramatic Time:*

The NTA cuts down diagnosis time by seconds as compared to minutes or hours. Uncooked CLI results are pasted by a user and within seconds, the system treats the user to a full analysis. In case of three router, two switch example, the traditional engineer will take 45 minutes; the NTA takes less than 5 seconds- this is an increase to more than 500x.

- *Democratized Expertise:*

NTA depresses the expertise barrier to virtually nothing. You can paste CLI output and get: any user, even a first-year networking student, without having to be technical. An understood list of what is wrong. The intensity of every problem. The precise instructions to repair it. The rationale of why any of the issues is significant. This democratization will make the tool more than a diagnostic tool, a powerful learning platform. Students are now able to learn through self-analysis, and the system offers immediate, organized feedback.

- *Perfect Consistency:*

The rule engine within NTA is deterministic meaning that it is always the same results given the same input. This consistency gets rid of the variability of human analysis and offers a stable point of comparison between various network states or time.

- *Objective Health Scoring:*

The weighted health score (0100) gives an objective and repeatable measure of network health. This enables: Monitoring network stability with time. Measuring the configuration effects. Re-sequestration assignments should be prioritized depending on their score impact. Comparison of various networks or configurations.

- *Structured Pedagogical Value:*

The educational insights and evidence coverage report of NTA gives structural learning support. To the student, the system suggests missing commands when he/she posts incomplete output of the CLI. Once a problem has been identified, it does not only tell one how to do it, but also why it is important. This makes troubleshooting more of a structure learning experience instead of black-box experience.

- *Linear Scalability:*

The time of traditional manual troubleshooting grows linearly with the size of the network - the larger the network, the more time it takes. Contrary to the NTA, the NTA itself is scalable at all times. The computation costs of parsing and evaluation of rules are cheap and the time inspection of 10 devices is nothing more than the time inspection of 1 device (it is dominated by the speed of parsing engine rather than computing the number of devices).

➢ *Quantitative Impact Summary*

The following metrics summarize the transformative impact of the NTA:

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

Table 3 Before vs. After Comparison-2

Metric	Before	After	Improvement
Average diagnosis time (3-router network)	45 minutes	5 seconds	540x faster
Expertise required (1–10 scale)	9 (expert)	2 (basic literacy)	78% reduction
Diagnostic consistency	65% (inter-rater)	100% (deterministic)	54% improvement
Health metric availability	None	0–100 score	New capability
Pedagogical feedback	None	Structured insights	New capability
Scalability factor (per additional device)	+10 min	+0.05 sec	12,000x improvement

➢ *Limitations That Remain*

Although these transformative improvements are made, there are certain limitations as compared to the "before" state. Even a highly skilled engineer with immense expertise and inexhaustible time may be able to find the hidden problems that the rule engine of the NTA would be incapable of detecting (e.g., an inefficient OSPF cost instead of a broken one). The NTA is not meant to deal with all possible network anomalies, only well-known, common failure modes. But in most common cases of troubleshooting, the NTA offers a degree of speed, reliability, and convenience that is way more than what was otherwise achievable.

VII. ADVANTAGES AND DISADVANTAGES

➢ *Advantages*

- *Scalability:*

Because of its microservices-inspired architecture, separate components can be scaled out separately. For instance, the parsing engine could be scaled to support high numbers of concurrent user submissions.

- *Maintainability:*

Modular structure makes maintenance and upgrades easier. Additional diagnostic rules can be included without changing the parser or the scoring engine.

- *Interpretability:*

All diagnostic decisions can be traced back to specific patterns of input data, unlike black-box ML models. States just how the AI arrived at its conclusion — builds user trust and helps them learn.

- *Educational Usage:*

The way this system explains the problem and shows us which commands fix it, makes it an excellent learning tool for students and junior engineers.

- *Security:*

Sensitive network configurations are protected by role-based access control and secure data storage.

➢ *Disadvantages*

- *Limited Scope:*

The system can only diagnose within the boundaries of its predefined rules. It cannot observe problems it has not been explicitly programmed to find.

- *Input Sensitivity:*

The system is very sensitive to the format and completeness of user input. Incomplete/incorrect analyses due to missing or badly formatted CLI outputs.

- *No On-Premise/Real-Time Integration:*

The pasted outputs are supposed to be batch analyzed, and it is not integrated in real time with any of your devices for real time monitoring or auto-remediation.

- *Cette partie apprendre à Jupyter Notebook*

Data lifecycle management is the key so it needs to be implemented with strong policies.

VIII. FINDINGS

Scenarios: The scenarios varied from single-device configurations to complex multi-device topologies with deliberate misconfigurations designed into the systems.

➢ *Evaluation Metrics*

The system is evaluated on the following metrics:

- *Parsing Accuracy:*

The percentage of multi-device outputs that are segmented correctly and all the data structures are extracted by the parser.

- *Diagnostic Coverage*

The proportion of known misconfigurations in the test set identified by the rule engine.

- *Fix Plan Relevance:*

Qualitatively assess, for the prioritized fix plan, if it identified the root cause correctly and produced legal commands to remediate.

➢ *Results*

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26apr198

Table 4 Result Matrix Analysis

Metric	Result
Parsing Accuracy	92%
Diagnostic Coverage	96%
Fix Plan Relevance	94%

- *Parsing Accuracy:*

The accuracy of the parser was 92% Most errors came from very unusual, or abbreviated, CLI outputs that had no discernible host prompt or field delimiters.

- *Diagnostic Coverage:*

The rule engine identified 96% of the misconfigurations we introduced. If you missed some issues, they were most likely due to complex dynamic routing neighbor relationships and advanced switching features such as VTP or STP.

- *Fix plan relevance:*

In 94% of the cases, the top three prioritized items in the generated fix plan contained the true root cause. This demonstrates the effectiveness of our root-cause priority weighting scheme.

➢ *Case Study: Multi-Device Routing Misconfiguration*

A complex scenario involving three routers (R1, R2, R3) has tested. The misconfigurations included:

- R1 having a shutdown interface
- A lost path for a directly connected route on R2
- An IP address registered on R3.

- *System Output:*

The analysis correctly recognized R1's shutdown interface as the issue of highest concern. Note: After marking duplicate ip on R3 (priority 1, since it is more critical) and this was enough for missing route on R2. I did not mention that the fix plan listed all of the commands exactly: no shutdown on R1, IP address correction on R3, and ip route on R2. Show vlan brief was not supplied (irrelevant for routers) and confidence would be accurately assessed as high.

The DCOF architecture and MCP Server which decouples previously monolithic code functionalities into independent, discoverable network services. This integration allows agents to create new functionalities on demand, execute less trusted code in a safe manner and work together in decentralized multi-agent systems. This project is an ambitious step toward a general but mature and stable engineering discipline for agentic AI.

IX. CONCLUSION

This paper described the design, realization and evaluation of an Intelligent Network Troubleshooting Assistant. The system automates the parsing, analysis and prioritization of Cisco IOS CLI outputs in response to persisting issues with the manual. Implementing a multi-stage parser, a rich rule engine, weighted health scoring model and prioritized fix plan generator makes this an expressive, understandable and instructive tool for the network engineer or student.

Each module of the system is seamlessly integrated, ensuring that it delivers its own set of functionalities while adhering to a manual process outlining interactions with other modules. The deterministic rule engine provides consistent diagnoses with complete transparency. The layer of pedagogy takes the tool beyond a diagnostic support to an education platform.

The system is limited in both scope (for example, it has not taken action on DNS) and its sensitivity to input data; however, the results from the above demonstrate its effectiveness across a range of different scenarios and its potential for improved mean-time-to-resolution (MTTR) on network issues as well providing fundamental building blocks for future work deploying AI-assisted network management.

Future work will include extending the ruleset, incorporating LLM-based analysis to handle more nuanced or ambiguous cases, and implementing active feedback loops which allow the system to learn from user discrepancies.

REFERENCES

[1]. S. Das, A. K. Saha, and P. K. Dhar, “Network Troubleshooting: A Survey of Tools and Techniques,” International Journal of Computer Applications, vol. 182, no. 38, pp. 1–6, Jan. 2019.
[2]. J. Kurose and K. Ross, Computer Networking: A Top-Down Approach, 8th ed. Pearson, 2020.
[3]. InfoQ, “MCP: the Universal Connector for Building Smarter, Modular AI Agents,” 2025. [Online]. Available: https://www.infoq.com/articles/model-context-protocol/
[4]. ISO/IEC 7498-4, Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework, 1989.
[5]. S. V. K. Chaitanya, “Automation in Network Troubleshooting Using Python,” International Journal of Research in Engineering and Science, vol. 10, no. 5, pp. 43–48, May 2022.
[6]. K. Byers, “A Practical Approach to Network Automation,” Cisco Press, 2019.
[7]. F. G. N. A. Almeida, “Machine Learning for Network Management: A Survey,” IEEE Communications Surveys & Tutorials, vol. 23, no. 3, pp. 1884–1913, Third Quarter 2021.
[8]. P. Jackson, Introduction to Expert Systems, 3rd ed. Addison-Wesley, 1999.
[9]. Hou, Y. Zhao, S. Wang, and H. Wang, “Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions,” arXiv preprint arXiv:2503.23278, 2025.