Project Details

1) Benchmarking Genius:

Problem Statement:

i) Telecom systems generate massive volumes of complex logs (KPMs, RCs, latency, etc.). Extracting insights manually takes time, delays troubleshooting, and can lead to missed issues.

ii) **Manual analysis** delays troubleshooting and risks missing critical issues. **Test run comparisons** require custom scripts, leading to inconsistency and inefficiency.

iii) **Optimization cycles are slow**, hindered by fragmented tools and human-dependent workflows. With **5G, Open RAN, and programmable infrastructure**, data volume and variability have exploded.

iv) Existing tools **stop at analysis**—they don't recommend or act on insights. **Key decisions** like app selection or configuration changes still rely on manual judgment.

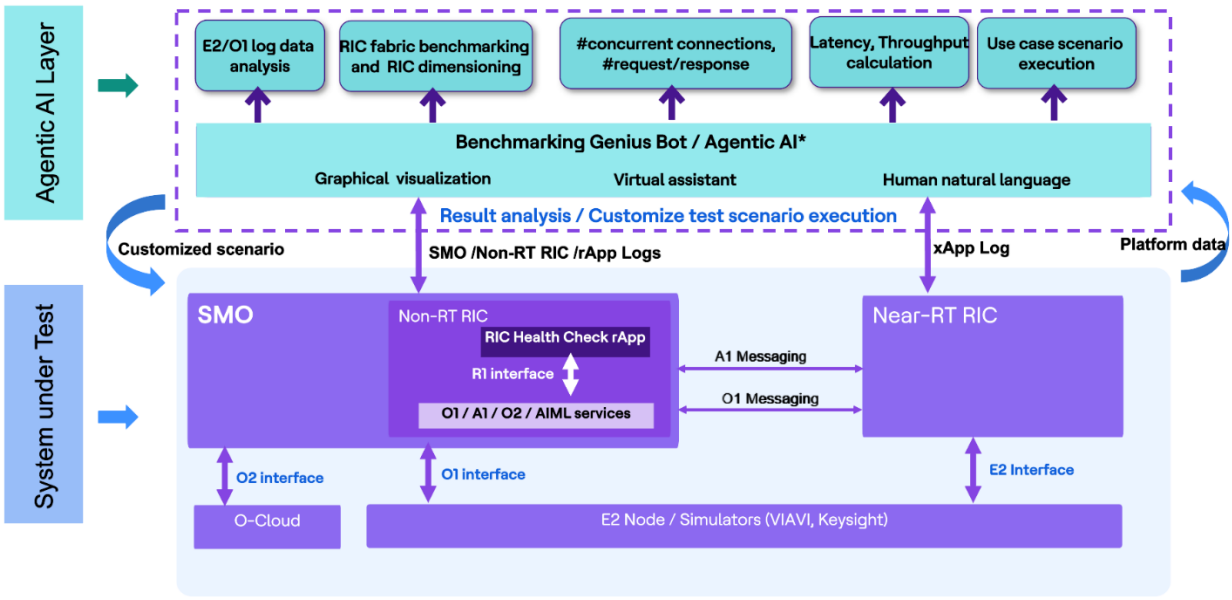
Solution:

The proposed solution uses a **GenAI** model combined with **CrewAI** to enable intelligent, autonomous behavior. It integrates with **RAN** components, **Prometheus** monitoring, and internal systems to continuously monitor performance and health. When issues arise, the AI can proactively suggest or trigger actions like launching specific applications or workflows. Users can also interact with it in natural language, making it a seamless, intelligent, and closed-loop system—all accessible through a simple **Flask-based interface.**

Log Analyzer Agent: Analyze logs & answer specific questions. Use tools to parse unstructured logs and store metrics (throughput, KPM/RC subscriptions, E2-latency, error lines, memory/CPU samples, etc.) in

pandas data frame. Based on user query it uses this data to give answers and summaries.

Execution Agent: Monitor resources & execute benchmarking xApp. Use SystemResourceTool (monitors CPU, Memory usage), BouncerFlowTool (Runs Benchmarking xApp)



Technology Used:

CrewAI, Flask, Dash, Plotly, Matplotlib, Kubernetes, Generative AI, Ollama, Gemini 2.5 Pro, Pandas, Python, Prometheus, HTML, CSS, Javascript

2) O-RAN Digital Twin:

Problem Statement:

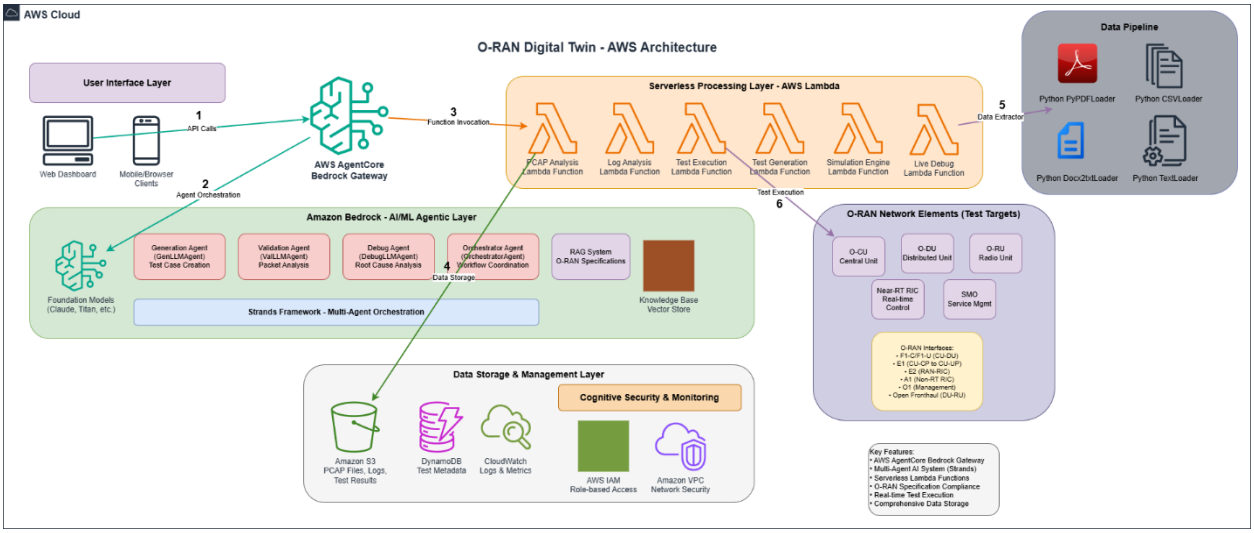
In today's multi-vendor RAN ecosystem, achieving production-grade maturity across interfaces such as Open Fronthaul, E1, F1, and E2 is increasingly difficult. Operators face significant challenges because protocol debugging, compliance checking, and message-level analysis demand deep expertise, which is often limited. Test case creation still

depends heavily on manually reading O-RAN and 3GPP specifications, while root-cause analysis requires labor-intensive packet inspection across multiple logs and PCAP files.
These manual processes slow down MVP deployments, reduce productivity, and increase operational overhead. This is exactly where the O-RAN Test Digital Twin helps—by using Agentic AI–driven test generation, automated validation, and intelligent debug agents to eliminate human-dependent tasks, accelerate testing cycles, and provide specification-aware analysis across multi-vendor environments.

Solution:

Digital Twin uses Agentic AI for Test Case Generation, Test Case execution, check interoperability, standard compliance, and security aspects. It uses four Agents:

Central Orchestrator Agent: Manages the lifecycle of testing, validation, and debugging using specialized agents (Gen-LLM, Val-LLM, Debug-LLM).
Generation Agent: Retrieves context from your knowledge base + web search and generates a final answer, while storing past Q&A in memory. Uses O-RAN WG Specifications, 3GPP specification, Vendor Specific documents and knowledge base.
Debugging Agents: Executes Test Cases based on user requirements and generate HTML logs. Debugs O-RAN components by pulling spec-based context, adding component-specific profile details, and reusing similar past cases from memory.
Validation Agent: Evaluates O-RAN test cases for compliance, retrieves specs, performs web-validated checks, and outputs a compliance score (0–100).



Technology Used:

AWS Cloud, AWS Strands, Generative AI, FastAPI, Python

Demo Video: https://youtu.be/r5TkhXmc2Kw?si=qVBYq-fhtPEzNBVu

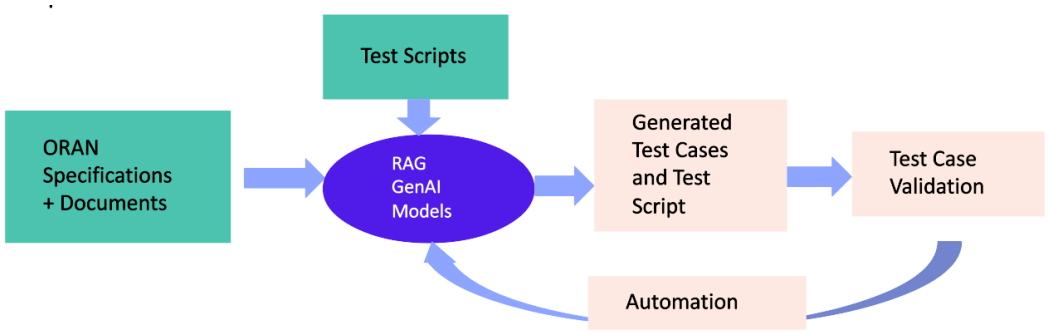
3) RAG Chatbot:

Problem Statement:

In the telecom domain, engineers deal with an overwhelming number of standards, specifications, and vendor documents. Identifying new test cases requires manually scanning through hundreds of pages across multiple O-RAN and 3GPP documents. After that, engineers must manually interpret the requirements and write corresponding test scripts for execution. This end-to-end process consumes significant effort, is slow, error-prone, and ultimately reduces productivity and delays validation cycles.

Solution:

I built a domain-specific RAG chatbot powered by the Qwen 2.5 Coder LLM, using FAISS as the vector database for efficient embedding storage. The system leverages hybrid retrieval—combining similarity search with BM25—to deliver highly accurate responses. The knowledge base is enriched with O-RAN Working Group specifications, 3GPP technical standards, and vendor-specific documentation to ensure telecom-grade precision. A FastAPI-based interface provides a seamless and interactive user experience.



Technology Used:

Langchain, Qwen 2.5 Coder, Ollama, Generative AI, FAISS Vector DB, FastAPI, Python

4) Retinal Image Analysis:

Problem Statement:

Ocular fundus diseases, including diabetic retinopathy (DR), age-related macular degeneration (AMD), retinal vein occlusion (RVO) and fundus tumors, impact millions of individuals globally. In developing nations, particularly in rural and remote regions where ophthalmic services are inadequate and ophthalmologists are sacre, early detection and timely referral for treatment may not be possible.

Significantly, fundus photography, which enables the fundamental detection of these diseases, is accessible and cost effective in the majority of global regions.

Solution:

In this project, we used Convolutional Neural Networks (CNNs) to create a multi-disease automatic detection platform that can classify 15 different types of common fundus diseases based on colour fundus photos. We build a deep learning platform that was trained, verified and tested using 2110 fundus photos from multiethnic data sets such as the RFMiD Dataset, 1000 Fundus images with 39 categories, and the STARE Dataset. Designed a few-shot learning model leveraging CNN for feature extraction and LSTM for sequential pattern recognition, enabling accurate detection with limited labeled data. Attained 84% accuracy with maximum recall of 95% and 77% precision.

Technology Used:

PyTorch, CNN, Python, Deep Learning

Github Link:

ssiddharth27/Few_Shot_Learning

5) Car Damage Detection:

Problem Statement:

Accurate car part segmentation and damage detection remain challenging tasks in computer vision due to the fine-grained nature of vehicle components, high intra-class similarity, diverse real-world conditions such as lighting, occlusion, and wear. Developing a robust system that can reliably identify and segment individual car parts is crucial for enabling automated vehicle inspection, repair cost estimation, and insurance claim processing.

Solution:

Developed an AI Powered vehicle damage detection system for an insurance client, automating inspection and streamlining claims processing. I designed an end-to-end pipeline for Car part segmentation, damage localization, color detection, and model recognition using advanced deep learning architectures such as Mask R-CNN and YOLO. The system achieved 95% accuracy in damage identification, significantly reducing manual effort and improving operational efficiency for large scale insurance workflows.

Technology Used:

Deep Learning, Mask R-CNN, YOLO, Python

Demo:

https://aspiring-heart-a11.notion.site/Work-Progress-a377a200e3ac48d38293a8d25336f4b8