Towards a Playground to Democratize Experimentation and Benchmarking of AI Agents for Network Troubleshooting

Towards a Playground to Democratize Experimentation and Benchmarking of AI Agents for Network Troubleshooting

Zhihao Wang
UESTC

Alessandro Cornacchia
KAUST

Franco Galante
Politecnico di Torino

Carlo Centofanti
University of L'Aquila

Alessio Sacco
Politecnico di Torino

Dingde Jiang
UESTC

Abstract

Artificial Intelligence (AI) and Large Language Models (LLMs), are increasingly finding application in network-related tasks, such as network configuration synthesis [22] and dialogue-based interfaces to network measurements [23], among others. In this preliminary work, we restrict our focus to the application of AI agents to network troubleshooting and elaborate on *the need for a standardized, reproducible, and open benchmarking platform, where to build and evaluate AI agents with low operational effort.* This platform primarily aims at standardize and democratize the experimentation with AI agents, by enabling researchers and practitioners — including non-domain experts such as ML/AI engineers— to evaluate AI agents on curated problem sets, without concerns for underlying operational complexities. We present a modular and extensible benchmarking framework that supports widely adopted network emulators [3, 18, 20, 21]. It targets an extensible set of network issues in diverse real-world scenarios – *e.g.*, data centers, access, WAN, etc. – and orchestrates the end-to-end evaluation workflows, including failure injection, telemetry instrumentation and collection, and agent performance evaluation. Agents can be easily connected through a single Application Programming Interface (API) to an emulation platform and rapidly evaluated. The code is publicly available at https://github.com/zhihao1998/LLM4NetLab.

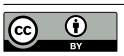
CCS Concepts

• **Networks** → **Network monitoring**; • **Computing methodologies** → **Natural language processing**.

Keywords

Network Troubleshooting, Large Language Models, AI Agents, Observability

ACM Reference Format:
Zhihao Wang, Alessandro Cornacchia, Franco Galante, Carlo Centofanti, Alessio Sacco, and Dingde Jiang. 2025. Towards a Playground to Democratize Experimentation and Benchmarking of AI Agents for Network Troubleshooting. In *1st Workshop on Next-Generation Network Observability (NGNO '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3748496.3748990



NGNO '25, Coimbra, Portugal

ACM ISBN 979-8-4007-2087-1/2025/09
https://doi.org/10.1145/3748496.3748990

1 Research Challenge

Given a network problem, network engineers need to undertake mechanical yet cumbersome steps to diagnose and mitigate the issue [23, 26]. They can be summarized as *(i)* identifying the right telemetry signals to collect, *(ii)* navigating dashboards and interpreting the collected data, *(iii)* taking corrective actions based on the insights derived from the telemetry data *(iv)* iterating on the previous speculating about root-causes and *what-if* hypothesis. Common examples include probing the network to explain packet drops [5, 26] or refining the detection logic – *e.g.*, request the collection of more fine-grained queue-length data to zoom into an ongoing incident [6], enable debug-level diagnostics, etc. This manual process is still complex, slow and error-prone, as it requires expert operators to reason across multiple dimensions.

Furthermore, modern network telemetry paradigms over programmable data planes – such as sketches [10, 16] and in-band network telemetry (INT) [17] – have expanded the range of available measurement strategies. The wide range of measurements introduces new degrees of freedom, but at the price of greater operational complexity. Thus, we observe that while programmable data planes and new telemetry techniques may have enhanced operators' visibility on the network, human intervention still remains a primary bottleneck in network triaging. For instance, it is the main obstacle to supporting "just-in-time" orchestration of network measurements – for example, dynamically firing or deactivating measurement tools based on live traffic conditions, performance anomalies, or evolving troubleshooting needs.

Thanks to their ability to parse multimodal data and, more recently, engage in natural-language-driven reasoning [2, 22, 23], LLMs – as a breakthrough category of AI models – hold a special promise for assisting network operators. As a result, our community has begun to explore how traditional approaches can evolve into more automated, LLM-assisted *intent-based* network monitoring and diagnosis solutions [1, 7, 13, 15].

Lack of holistic platforms and benchmarks. Nevertheless, existing experimentation environments [3, 20] are often limited in scope, lacking standardized and reproducible benchmarks. Recently, benchmarks for LLM agents have been proposed in the area of network configuration, such as NetConfEval [22]. This line of work focuses on the evaluation of non-agentic AI on static benchmarks, where one-shot, offline executions suffice. While this setting is adequate for various problem instances in network configuration, *network troubleshooting* is an inherently more dynamic and interactive problem space. It requires real-time feedback loops with the network, where AI agents must not only observe but also probe, react, and refine based on the evolving system conditions. Thus, it is

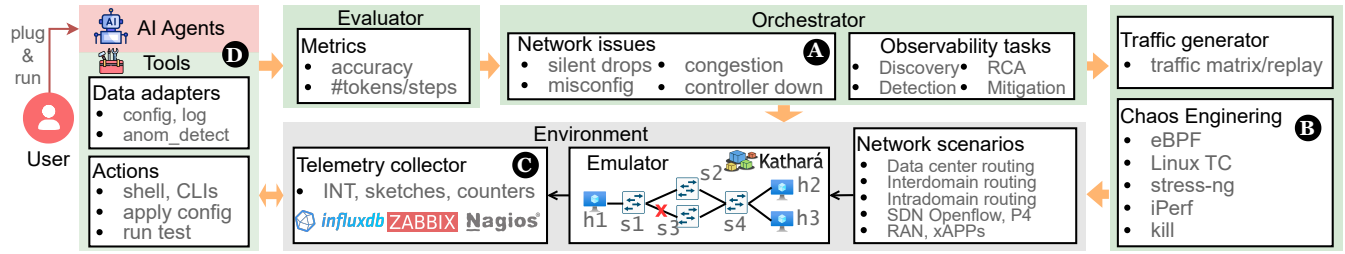


Figure 1: Architecture of the proposed framework.

essential to evaluate AI agents in environments that allow for such interactive, closed-loop operations, where they can dynamically adapt their strategies based on real-time telemetry and network state. Unfortunately, as of today, engineers wishing to experiment with AI agents for network troubleshooting must still shoulder the significant burden of developing custom infrastructure code to enable this agent-environment interaction. No modular, low-effort platform – abstracting network environments (e.g., cloud, edge, mobile, satellite), protocols, failure modes, and tooling – exists to date.

Our vision. We draw attention to the risk that, in the absence of standardized evaluation practices, research in this area remains fragmented and difficult to compare. In response, we argue for a holistic, universally accepted platform that combines an extensible benchmark suite and a flexible, plug-and-play network playground – e.g., based on existing network emulators. We develop a preliminary Proof-of-Concept (PoC) and validate it with a ReAct agent for a toy-case failure scenario. We then outline challenges and future directions towards our vision.

2 Proof-of-Concept

An overview of our playground's architecture is shown in Figure 1. The playground targets users (e.g., ML engineer) willing to implement an AI agent for network troubleshooting and helps them to connect it to the network environment. We provide an over-writable callback to implement the AI agent logic – e.g., `execute_agent` and simplify its evaluation as follows. The user can task the AI agent with one of the predefined network issues and scenarios (A), which together constitute the evaluation benchmark. We offer a yaml-based declarative interface to express this choice. The user can kickoff the experiment workflow in a single command. Then, our platform is responsible for instantiating the experiment and orchestrating the underlying operations such as traffic generation, fault (or misconfiguration) injection (B), telemetry instrumentation and telemetry collection (C). The agent can then interact with the network environment through a set of MCP-based [14] tools (D) that expand its capabilities. Our design follows related work [19] in microservice applications.

Based on this design, we prototype an initial PoC on top of Kathara [3] and validate an end-to-end network failure scenario, which we triage using a ReAct [25] AI agent. In our toy-case example, a `DeepSeek-R1-0528` agent tries to triage a network issue, whose root cause is a lossy link. The network topology comprises four BMv2 switches and is shown in Fig. 1 (emulator inset). We

User: You are an expert networking engineer who has been tasked with detecting anomalies in a deployed network topology: s1:'name': 'eth0', 'port': 1, 'connected_to': 'h1', 'connected_port': 1[…].You will begin by analyzing the network's state, and then submit your solution:[…] You are provided with the following APIs to interact with the network: [..]
ReAct Agent: ***Thought:*** To begin anomaly detection, I'll first check the overall network reachability […] ***Action:*** `test_reachability()`
Framework: 'h1 ping h2: 10 packets transmitted, 10 received, 0% packet loss', 'h1 ping h3: 10 packets transmitted, 0 received, 100% packet loss' […]
ReAct Agent: ***Thought:*** […] the problem may lie in the s1-s3 link or the s3-s4 link. I'll start by checking port counters on s1's port connected to s3 (port 3) to detect potential packet loss. ***Action:*** `bmv2_get_counters("s1")`
Framework: 'MyIngress.ingress_port_counter', 'MyEgress.egress_port_counter'
ReAct Agent: ***Thought:*** […] Since we're investigating the path from h1 to h3, which should go through s1's port 3 (egress to s3), I'll check the egress counter for port 3 to see if packets are being sent out properly. ***Action:*** `bmv2_counter_read("s1", "MyEgress.egress_port_counter", 3)`
Framework: MyIngress.ingress_port_counter[3]= (980 bytes, 10 packets)
The agent checks more counters
ReAct Agent: ***Thought:*** […] suggest a potential unidirectional link failure or port malfunction specifically affecting traffic from s3 to s1.[…] ***Action:*** `submit_findings()`

Category	API	Description
Data Adapters	`get_switch_logs/info`	Get device running logs/information
	`ovs/bmv2_dump_ports`	Show all ports of OVS/Bmv2 P4 switch
	`bmv2_get_counters`	Get counters in a BMv2 P4 switch
	`bmv2_counter_read`	Read counter values in a BMv2 P4 switch
	`get_topology`	Obtain structured topology information
Network Operators	`config_frr_bgp/ospf`	Configure BGP/OSPF in FRRouting
	`ovs_table_add/modify`	Add/modify flow table entry of OVS
	`bmv2_table_add/modify`	Add/modify table entry in BMv2 P4 switch
	`test_reachability`	Check reachability between all hosts

Figure 2: Agent reasoning and tools implemented in the PoC.

inject an artificial packet loss issue on the s1 → s3 link. We task the agent with (1) detecting and (2) localizing the anomaly. Fig. 2 (top) illustrates the agent's reasoning trajectory, overall consisting of 15 steps. The agent is prompted with the operator's intent (User line) and is given no clue about the anomaly root-cause. It can access the tools illustrated in Fig. 2 (bottom) to interface with the environment. It begins with active probing via `get_reachability()`, detects loss between `h1` and `h3`, then proceeds to querying port counters using `bmv2_counter_read()`. Based on the retrieved statistics, it successfully localizes the fault to s3.

3 Future agenda

Looking ahead, we outline the following key directions which are part of our current and future work.

Benchmark curation. We aim to curate a diverse benchmark of failure scenarios, spanning heterogeneous networks (e.g., data centers vs. geographical networks) network stacks and failure type. Each scenario is manually constructed with defined triggers, observability signals (e.g., INT latency spikes, counter anomalies) and root

causes. A primary challenge in curating such a diverse benchmark lies in minimizing human effort while ensuring sufficient scenario *variations*. We plan to study how to automate the generation of these variations, starting from a well-defined subset of network issues across different domains. To this end, we can draw inspiration from similar approaches in software engineering [8]. Furthermore, we plan to explore automatic ways of tuning the level of complexity of the injected problem sets, e.g., by tweaking temporal patterns or combining multiple failures. It can be accomplished via parametric failure injection templates [4], or, for other class of failure modes, we could explore LLMs themselves to generate failure modes by reasoning on configuration files and network setups.

Agent–environment interfaces. We envision developing unified agent–environment interfaces that abstract low-level complexity and expose structured access to both telemetry (e.g., system metrics, INT, sketches) and control (e.g., configuration updates, active probing). We aim to align these interfaces with MCP to support structured context exchange and standardized agent–environment interaction. Prior work has shown that equipping AI agents with task-specific tools is critical for diagnostics like RCA [24]. Rather than parsing raw, heterogeneous telemetry directly with AI agents, agents invoke and interpret outputs from modular tooling, such as ML-based anomaly detectors. This design mitigates the limitations of AI agents in handling domain-specific tasks and enables more robust, composable reasoning pipelines.

Automated assessment of agent behavior. The results generated by LLMs are in the form of unstructured natural language reasoning trajectories (Fig. 2). Application developers mostly resort to manual inspection to assess the correctness, accuracy and efficiency of the reasoning process. The process is time-consuming, limits experiments scalability, and obstructs fairness and reproducibility. To address the gap, we propose extending our framework with automated behavioral checkups – e.g., leveraging *LLM-as-a-judge* [12] – to evaluate agent trajectories in a more structured and holistic manner. If LLM-as-as-judge is used, open research questions include how to structure the judge agent (e.g., single-agent vs multi-agent architecture), and whether or to fine-tune or not the underlying model. Emerging observability tools for agentic AI applications [9, 11] enable rich logging, tracing and monitoring of agent executions and we plan their integration with the platform to provide out-of-the-box visibility into agent behavior.

References

[1] Antonino Angi, Alessio Sacco, and Guido Marchetto. 2025. LLNet: An Intent-Driven Approach to Instructing Softwarized Network Devices Using a Small Language Model. *IEEE Trans. Netw. Serv. Manag.* (2025).

[2] Kaan Aykurt, Andreas Blenk, and Wolfgang Kellerer. 2024. NetLLMBench: A Benchmark Framework for Large Language Models in Network Configuration Tasks. In *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN'24)*. 1–6.

[3] Gaetano Bonofiglio, Veronica Iovinella, Gabriele Lospoto, and Giuseppe Di Battista. 2018. Kathará: A container-based framework for implementing network function virtualization and software defined networks. In *Proceedings of 2018 IEEE/IFIP Network Operations and Management Symposium (NOMS'18)*. 1–9.

[4] Cloud Native Computing Foundation. 2025. Chaos Mesh: A Cloud Native Chaos Engineering Platform. https://chaos-mesh.org Accessed: 2025-05-30.

[5] Kaihui Gao, Chen Sun, Shuai Wang, Dan Li, Yu Zhou, Hongqiang Harry Liu, Lingjun Zhu, and Ming Zhang. 2022. Buffer-based End-to-end Request Event Monitoring in the Cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, 829–843.

[6] Fengchen Gong, Divya Raghunathan, Aarti Gupta, and Maria Apostolaki. 2024. Zoom2Net: Constrained Network Telemetry Imputation. In *Proceedings of the ACM SIGCOMM 2024 Conference*. Association for Computing Machinery, 764–777.

[7] Md Arafat Habib, Pedro Enrique Iturria Rivera, Yigit Ozcan, Medhat Elsayed, Majid Bavand, Raimundus Gaigalas, and Melike Erol-Kantarci. 2025. Llm-based intent processing and network optimization using attention-based hierarchical reinforcement learning. In *Proceedings of 2025 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 1–6.

[8] Naman Jain, Manish Shetty, Tianjun Zhang, King Han, Koushik Sen, and Ion Stoica. 2024. R2E: turning any GitHub repository into a programming agent environment. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*. JMLR.org, 21196–21224.

[9] LangFuse. 2025. Open Source LLM Engineering Platform. https://langfuse.com. Accessed: 2025-07-01.

[10] Jonatan Langlet, Ran Ben Basat, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. 2023. Direct Telemetry Access. In *Proceedings of the ACM SIGCOMM 2023 Conference*. Association for Computing Machinery, 832–849.

[11] LangSmith. 2025. Ship agents with confidence. https://www.langchain.com/langsmith. Accessed: 2025-07-01.

[12] Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. 2025. From Generation to Judgment: Opportunities and Challenges of LLM-as-a-judge. arXiv:2411.16594 [cs.AI] https://arxiv.org/abs/2411.16594

[13] Yuanpeng Li, Zhen Xu, Zongwei Lv, Yannan Hu, Yong Cui, and Tong Yang. 2025. LLM-Sketch: Enhancing Network Sketches with LLM. arXiv:2502.07495

[14] Model Context Protocol Project. 2025. Model Context Protocol: Introduction. https://modelcontextprotocol.io/introduction. Accessed: 2025-05-31.

[15] Seyed Mohamad Moghadas, Yangxintong Lyu, Bruno Cornelis, Alexandre Alahi, and Adrian Munteanu. 2025. Strada-LLM: Graph LLM for traffic prediction. arXiv:2410.20856

[16] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. 2023. Sketchovsky: Enabling Ensembles of Sketches on Programmable Switches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, 1273–1292.

[17] P4.org. 2020. P4 In-band Network Telemetry (INT) Specification. https://p4.org/p4-spec/docs/INT_v2_1.pdf. Accessed: 2025-05-30.

[18] M. Peuster, H. Karl, and S. van Rossem. 2016. MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments. In *Proceedings of 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN'16)*. 148–153.

[19] Manish Shetty, Yinfang Chen, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Las-Casas, Shachee Mishra Gupta, Suman Nath, Chetan Bansal, and Saravan Rajmohan. 2024. Building AI Agents for Autonomous Clouds: Challenges and Design Principles. In *Proceedings of the 2024 ACM Symposium on Cloud Computing (SoCC '24)*. Association for Computing Machinery, 99–110.

[20] SRL-Labs. 2020. Containerlab: Container-Based Networking Labs. https://containerlab.dev/. Accessed: 2025-05-30.

[21] Mininet Team. 2011. Mininet: An Instant Virtual Network on your Laptop (or other PC). https://mininet.org/. Accessed: 2025-05-30.

[22] Changjie Wang, Mariano Scazzariello, Alireza Farshin, Simone Ferlin, Dejan Kostić, and Marco Chiesa. 2024. NetConfEval: Can LLMs Facilitate Network Configuration? *Proc. ACM Netw.* 2, CoNEXT2 (2024).

[23] Haopei Wang, Anubhavnidhi Abhashkumar, Changyu Lin, Tianrong Zhang, Xiaoming Gu, Ning Ma, Chang Wu, Songlin Liu, Wei Zhou, Yongbin Dong, et al. 2024. NetAssistant: Dialogue based network diagnosis in data center networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2011–2024.

[24] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2024. RCAgent: Cloud Root Cause Analysis by Autonomous Agents with Tool-Augmented Large Language Models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24)*. Association for Computing Machinery, 4966–4974.

[25] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR'23)*.

[26] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, Pengcheng Zhang, Dennis Cai, Ming Zhang, and Mingwei Xu. 2020. Flow Event Telemetry on Programmable Data Plane. In *Proceedings of the ACM SIGCOMM 2020 Conference*. Association for Computing Machinery, 76–89.