NETMEND: CAN LLM AGENTS REPAIR FAULTS IN LIVE NETWORKS?

Anonymous authors
Paper under double-blind review

ABSTRACT

As LLM agents move into infrastructure operations, rigorous evaluation of their network-troubleshooting ability becomes critical. However, most existing benchmarks evaluate diagnosis rather than repair. Others evaluate repair, but in simplified settings far removed from operational networks. We present **NetMend**, a benchmark that evaluates whether LLM agents can repair injected faults in live containerized networks. The agent starts from a symptom report with the injected fault hidden, and must localize and repair it through typed tools whose actions are rendered into real operator commands. Correctness is judged from the state of the running network, rather than from the agent's own report. NetMend also scores whether the agent found the true root cause and whether its fix damaged unrelated parts of the network. Underlying NetMend is a task generator rather than a fixed dataset. It synthesizes networks across blueprints, routing protocols, and network operating systems. Each task is emitted as a replayable specification with a computed difficulty, and six fault classes carry a reference solution. Beyond synthetic networks, NetMend also rebuilds two production topologies from real vendor configurations. NetMend currently comprises 29 fault classes and their compound combinations, instantiated on networks of configurable scale. Across seven open-weight models, repair success ranges from 27.6% to 94.3%, and even successful agents take three to five times as many tool calls as the reference solution. About one in nine correctly localized faults is still not repaired, whereas only one repair in fifty succeeds without the correct root cause. Code is available at `https://anonymous.4open.science/r/NetMend-0517`.

1 INTRODUCTION

As LLM agents advance from language tasks to operating real systems, they are gradually deployed in production network operations (Wang et al., 2025a; Asadli et al., 2026). Network failures are a leading cause of service outages, and resolving them remains a time-critical and expert-driven process (Protogeros et al., 2026). However, automating this process poses substantial challenges. The agent must gather evidence over multiple steps. Each step yields raw and often ambiguous command output. Even then, partial observability and a large space of candidate causes complicate root-cause inference. Besides, repair introduces additional risk. An incorrect change takes effect on the live network and sometimes can disrupt traffic beyond the original fault. Therefore, safe deployment requires rigorous evaluation of whether an agent can actually repair a faulty network.

However, significant gaps persist in existing evaluations. First, most benchmarks evaluate diagnosis rather than repair. The agent reports a suspected root cause, and the report is compared against a ground-truth label (Wang et al., 2025d; Yang & Xu, 2026; Wang et al., 2025a; Cai et al., 2026). Repair is typically excluded because the correctness and safety of a proposed remedy are difficult to assess (Wang et al., 2025d; Asadli et al., 2026). Consequently, these benchmarks capture an agent's ability to identify a fault, but not its ability to restore the network. Second, the benchmarks that do evaluate repair operate only in simplified environments. In NetArena, agents repair broken links and forwarding rules in Mininet, where no routing software runs (Zhou et al., 2026). In Cornetto, agents repair static configuration files, and the result is verified formally rather than on a running network (Protogeros et al., 2026; Asadli et al., 2026). In cloud operations, agents mitigate incidents at the service layer of microservice deployments (Chen et al., 2025; Jha et al., 2025; Zhang et al., 2025). Thus, no existing benchmark evaluates whether an agent can localize and repair faults in a live

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

network running real routing protocols. Table 1 positions NetMend against these benchmarks along eight capabilities of the environment, the verdict, and the tasks. No existing benchmark combines a real network operating system with a graded repair, and none scores collateral damage on a live network.

Table 1: Capability comparison with existing network-agent benchmarks. **Live**: acts on a running network. **Real NOS**: real routing software. **Repair**: the fix is graded. **State**: verdict from system state. **RCA**: root cause scored. **No-Dmg.**: collateral damage scored on the live network. **Gen.**: generated tasks. **Ref.**: per-task reference solution (NetMend: six of 29 classes).

Benchmark	Live	Real NOS	Repair	State	RCA	No-Dmg.	Gen.	Ref.
NIKA	✓	✓	×	×	✓	×	×	×
NetOpsBench	✓	✓	×	✓	✓	×	✓	×
NetArena	✓	×	✓	✓	×	×	✓	×
Cornetto	×	×	✓	×	×	×	✓	×
NetMend (ours)	✓	✓	✓	✓	✓	✓	✓	✓

To close this gap, we present **NetMend**. NetMend injects faults into live containerized networks and requires an LLM agent to localize and repair them. Each network is built with containerlab (Nokia and contributors, 2024), and every device runs real routing software (FRRouting Project, 2024). The agent receives the network's design documentation and a symptom report, while the injected fault remains hidden. It must localize and repair the fault through typed inspection and repair tools, each of which renders its arguments into a real operator command. Correctness is then judged from the state of the running network, using reachability probes, device state, and routing tables, rather than from the agent's own report. NetMend also scores whether the agent identified the true root cause and whether its changes damaged unrelated parts of the network.

Underlying NetMend is a task generator rather than a fixed dataset. A network generator synthesizes networks across blueprints, routing protocols, and network operating systems, and a fault injector turns each network into a task. Each task is emitted as a replayable specification that records how to rebuild the network and re-inject the fault. A computed difficulty accompanies each specification, and six fault classes carry a reference solution. Single faults can be composed into compound tasks with multiple independent root causes. The current release covers 29 fault classes, with network scale controlled by the generator.

Our main contributions are summarized as follows:

- **A live-network repair benchmark.** We introduce NetMend, to our knowledge the first benchmark in which an LLM agent repairs faults in live networks running real routing software. NetMend automates the entire cycle from network generation and fault injection to repair verification. Success is determined solely by the network state, making the evaluation robust to agent hallucination.
- **A heterogeneous-network generation pipeline.** We develop an automated network generator that constructs diverse networks in which devices from different vendors interoperate, and we rebuild two production topologies from real vendor configurations.
- **A multi-dimensional evaluation scheme.** We evaluate each repair by correctness, safety, and efficiency. Correctness covers network recovery and root-cause identification. Safety penalizes repairs that break reachability elsewhere in the network. Efficiency is measured against a per-task reference solution where one is defined and by the steps and tokens consumed.

2 Related Work

LLM agents for network operations. Before LLMs, fault localization and mitigation were automated with statistical models and predefined action sets (Arzani et al., 2018; Harsh et al., 2023a;b; Gao et al., 2020; Levy et al., 2020; Namyar et al., 2025). LLM-based systems are now deployed in production networks. NetAssistant answers operators' diagnosis queries through dialogue (Wang et al., 2024b), BiAn ranks suspect devices during incident investigation (Wang et al., 2025a), RCACopilot drafts root-cause analyses for cloud incidents (Chen et al., 2024), Confucius coordinates multi-agent management workflows (Wang et al., 2025b), and SADE escalates troubleshooting by symptom (Tseng et al., 2026). These systems stop at localization or select among fixed mitigations, and each is evaluated on internal production data, so their results are neither comparable across systems nor reproducible outside them.

Benchmarks for network agents. Existing benchmarks progress from static knowledge tests (Maatouk et al., 2026; Liu et al., 2025) and configuration synthesis (Mondal et al., 2023; Mani et al., 2023; Wu et al., 2024; Wang et al., 2024a; Twabi et al., 2026) to interactive diagnosis. NIKA and NetOpsBench score detection, localization, and root-cause identification on emulated networks (Wang et al., 2025d;c; Yang & Xu, 2026). Their reports are compared against labels, and mitigation is explicitly deferred (Wang et al., 2025d). Benchmarks that do grade repair remain simplified. In NetArena, agents repair broken links and forwarding rules in a five-node Mininet topology, where no routing software runs, and collateral damage is checked by a binary safety test rather than measured on network state (Zhou et al., 2026). In Cornetto, agents repair static configurations under formal verification (Fogel et al., 2015; Beckett et al., 2017), without a running network, and regressions are checked formally in the same way (Protogeros et al., 2026; Asadli et al., 2026). Table 3 in the appendix compares these environments in detail. Outside networking, SWE-bench and other environment-grounded agent benchmarks established the principle of grading the artifact rather than the claim (Jimenez et al., 2024; Zheng et al., 2026; Yao et al., 2024; Barres et al., 2025; Liu et al., 2024; Zhou et al., 2024; Mialon et al., 2024; Xie et al., 2024). NetMend applies this principle to network operations, where the artifact is the running network.

3 The NetMend Benchmark

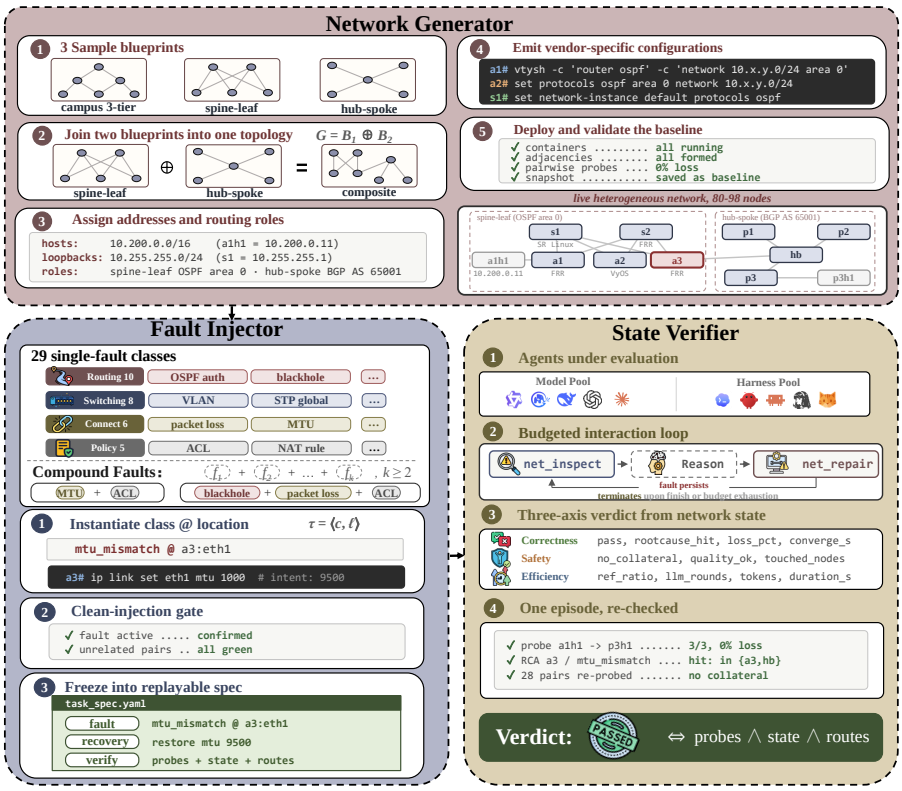


Figure 1: Overview of NetMend. The network generator builds a live containerized network from a blueprint. The fault injector turns the network into a replayable repair task. The state verifier scores the agent's repair from the state of the running network.

Figure 1 shows the three components of the benchmark. The network generator synthesizes a heterogeneous network and validates its fault-free state (Section 3.1). The fault injector instantiates a fault class, confirms its effect, and emits a replayable task specification (Section 3.2). The state verifier scores the agent's repair from the resulting network state (Section 3.3).

Let G denote a generated network and $\mathcal{C}$ the set of 29 fault classes. For a class $c \in \mathcal{C}$, $L_G(c)$ denotes the locations in G at which c can be instantiated, such as an interface, a device, or a protocol session.

A task is defined as the instantiation of one class at one location:

$$\tau = \langle G, c, \ell \rangle, \qquad c \in \mathcal{C},\ \ell \in L_G(c). \tag{1}$$

Each task is stored as a replayable specification with three parts:

$$S(\tau) = \big(S_{\text{net}},\, S_{\text{fault}},\, S_{\text{ver}}\big), \qquad S_{\text{ver}} = (P, D, R), \tag{2}$$

where S_{net} contains the blueprint, the protocol assignment, and the per-device configurations of G. The fault specification S_{fault} contains the injection command for c at ℓ and the recovery command. The verification targets S_{ver} contain the reachability probes P, the device-state checks D, and the routing-table entries R. The generator produces S_{net}, the injector adds S_{fault} and S_{ver}, and the verifier evaluates S_{ver} once the agent has finished. Section 3.4 describes the quality controls that every task passes before release.

3.1 Network Generator

Topology. The generator samples a topology from three blueprints, a three-tier campus, a spine–leaf fabric, and a hub–spoke wide-area network. Two blueprints can be joined into a composite topology, $G = B_1 \oplus B_2$, so that an OSPF fabric and a BGP wide-area segment coexist in one network.

Addressing and configuration. The generator assigns a host prefix to every access segment, a loopback address to every router, and a protocol role to every segment, such as an OSPF area or a BGP autonomous system. The design is then rendered in each device's vendor syntax, for example `vtysh` commands for FRR, `set` statements for VyOS, and network-instance configuration for SR Linux.

Deployment and validation. The network is deployed with containerlab (Nokia and contributors, 2024). A validation pass confirms that every container is running, every routing adjacency has formed, and every probe pair passes without loss. The validated state is stored as the baseline snapshot.

Coverage. Generated networks span eight routing protocols (OSPF, RIP, BGP, iBGP, IS–IS, static routing, MPLS L3VPN, and SRv6) and three network operating systems (FRR, VyOS, and SR Linux). The vendor is a per-segment parameter, so a fault can be placed on any vendor while the rest of the network stays fixed. Beyond synthetic networks, two production topologies are rebuilt from real vendor configurations and carry the same task format. Network scale is a generator parameter, and Section 4.4 varies it from about 25 to 180 nodes.

3.2 Fault Injector

Fault classes. The injector draws from 29 single-fault classes in four categories, routing (10 classes), switching (8), connectivity (6), and policy (5). Table 5 in the appendix lists every class with the device-level change it injects and its computed difficulty. Single faults can be composed into compound tasks of any order $k \geq 2$, each with multiple independent root causes. A compound task places $k \geq 2$ single faults in the same network, $\tau = \langle G, \{(c_i, \ell_i)\}_{i=1}^{k} \rangle$, where each pair (c_i, ℓ_i) is one class at one location as in Equation 1. Components are assigned to pairwise distinct devices, excluding the root-cause and victim devices of components already placed, which prevents interference between components and leakage of one component's root cause through the merged symptom. Each component is verified independently, and a compound episode passes only if every component is repaired. The current release contains two compound classes with $k = 2$.

Instantiation and gate. The injector instantiates a class c at a concrete location ℓ as a real command executed on the target device in the manner of chaos engineering (Basiri et al., 2016), such as lowering an interface MTU, and records the intended value alongside.

A gate confirms that the fault is active and re-probes all baseline pairs unrelated to it, so damage outside the intended fault region is detected before the agent starts.

Specification. The task is frozen into the specification $S(\tau)$ of Equation 2. The injector writes S_{fault}, the injected change and its recovery action. It also writes S_{ver}, since the injected class and location determine which targets certify a repair. These targets are thus fixed before the agent acts, and the verifier later evaluates them as written. Rebuilding the network from S_{net} and re-injecting the fault from S_{fault} reproduces the task exactly.

3.3 State Verifier

Agents and interaction loop. An episode is a single run of a tool-calling agent on an injected task. Its input consists of the design documentation of the network, which specifies the topology, the addressing plan, and the protocol role of every device, and a symptom report in audited wording (Section 3.4). The injected fault is withheld. Two typed tools give the agent access to the network. `net_inspect` provides twenty read-only views, such as probe results, interface detail, routing tables, and protocol-neighbor state. `net_repair` provides twenty-three write actions, such as restoring an interface MTU or removing a wrong static route. The agent chooses an action and its arguments, and the tool renders and executes the operator command in the device's vendor syntax; the action space is closed (Appendix B), and each call consumes one unit of the tool budget β. The loop ends at step $T = \min(t_{\text{fin}}, \beta)$, where t_{fin} is the step at which the agent calls a finish action that names the root-cause device $\hat{\ell}$ and the fault type $\hat{c}$. Figures 7 to 9 in the appendix show one complete episode.

Correctness, Safety, Efficiency Each repair is evaluated along correctness, safety, and efficiency, and every verdict is derived from the realtime running network. Let N_T denote the state of the network when the episode ends, and let $V_c \subseteq S_{\text{ver}}$ be the subset of targets that class c requires. An episode passes if all required targets hold on the final state, which extends to compound tasks by requiring every component to pass:

$$\text{pass}(\tau) = \begin{cases} \bigwedge_{v \in V_c} v(N_T), & \tau = \langle G, c, \ell \rangle, \\ \bigwedge_{i=1}^{k} \text{pass}(\tau_i), & \tau = \langle G, \{(c_i, \ell_i)\}_{i=1}^{k} \rangle, \end{cases} \tag{3}$$

where $\tau_i = \langle G, c_i, \ell_i \rangle$ is the i-th component.

Correctness. After the agent finishes, the verifier re-runs the verification targets stored in the specification. A probe target holds when the reachability probe succeeds, a state target when the device attribute equals its intended value, and a route target when the routing table contains the expected entry. The evidence set V_c is fixed per fault class. Redundant topologies reroute traffic around a broken adjacency, so a probe can succeed while the fault persists. Protocol faults are therefore judged on device state, such as an OSPF adjacency returning to Full or a BGP session re-establishing. Route faults are judged on the routing table, including the origin of an entry. Several classes require both a state change and a passing probe, since either alone can be produced by a wrong fix. Two rules apply throughout. A measurement that cannot be read counts as unrepaired, so measurement error never awards credit. The predicate is re-polled after the repair, and the time to its first pass is recorded as the convergence time, $t_{\text{conv}} = \min\{t \geq t_{\text{fin}} : \text{pass}(N_t)\}$. Root-cause identification is scored in addition. Since some faults admit several equivalent repair points, the injected location ℓ is expanded to its equivalence class $E(\ell)$. Let M denote the set of devices on which the agent issued a repair action. The root cause counts as identified when the declared node lies in $E(\ell)$ or a repair action was issued on a node in $E(\ell)$, $\text{hit}(\tau) = \mathbf{1}[\hat{\ell} \in E(\ell) \vee M \cap E(\ell) \neq \emptyset]$. An episode that ends without a declaration therefore earns the credit only through its repair actions.

Safety. Let P_0 be the probe pairs recorded in the baseline snapshot and $P_\Delta \subseteq P_0$ the pairs whose endpoints or path touch a device the agent modified. The verifier re-probes every pair in P_Δ and grants the no-damage credit only when all of them remain reachable:

$$\text{no_collateral}(\tau) = \bigwedge_{p \in P_\Delta} \text{reach}_{N_T}(p). \tag{4}$$

Pairs that remain reachable are further compared with their baseline path for hop count and round-trip time (Appendix D). Both checks are defined only on repaired episodes, because in an unrepaired network the fault itself keeps its pairs down, and their rates are therefore reported together with the number of repaired episodes.

Efficiency. Tasks of six classes (Section 4.1) carry a reference solution, an idealized lower bound on the trajectory of an agent that does not know where the fault is. Let C denote the devices and links on the forwarding path between the probe endpoints. The reference localizes the fault by binary search over C, repairs it, and verifies the repair, so its length is $|A^*| = \lceil \log_2 |C| \rceil + 2$ tool calls; the bound assumes that every inspection halves the candidate set, which no real inspection guarantees, so ratios above one measure distance from an ideal rather than waste. A second derivation expands the known recovery action of the class into fixed localization, repair, and verification steps and serves as a check that no trajectory can be shorter than the class allows. Let A denote the tool calls of a trajectory. A solved episode is scored by $\mathrm{ref_ratio} = |A|/|A^*|$, and the ratio is undefined on unsolved episodes. The steps, tokens, and wall-clock time of every episode are logged.

3.4 Task Quality and Statistics

Symptom auditing. The symptom report is the agent's main source of fault information, and careless wording reveals the answer. Each symptom is written at two levels, a plain level that names neither the fault type nor the location and a hinted level that reveals the fault type for 27 of the 29 classes and the faulty node for 16. Evaluation uses the plain level, whose wording is verified to contain no fault-type and no location terms, and the hinted level serves as a control. Appendix G quantifies the inflation that unaudited wording introduces.

Computed difficulty. Each task carries a difficulty score computed from five ordinal factors of its fault class, the network layer, the observability, the diagnostic and recovery complexity, and the impact scope, and binned into easy, medium, and hard bands. Appendix B gives the weights and thresholds, which are fixed constants of the release, and Appendix E reports how far the score predicts measured outcomes.

Statistics. Each of the 29 fault classes is hosted by one of four network profiles, a routed campus with OSPF and IS–IS (19 classes), a dual-gateway campus with BGP and static routing (6), a VXLAN overlay over BGP (2), and an MPLS L3VPN (2), and every profile is generated at three seeds. Across 800 sampled generator profiles, network size ranges from 80 to 98 nodes with a mean of 90.1, and the twelve networks of the main grid contain 84 to 98 nodes.

4 Experiments

4.1 Setup

Tasks. The main evaluation instantiates each of the 29 single-fault classes on three generated networks, one per generator seed, for 87 tasks. The networks contain 84 to 98 nodes and run FRR (FRRouting Project, 2024) on every device, since its memory footprint of 32 MB per node, against 218 MB for VyOS and 1.2 GB for SR Linux, keeps the evaluation within the compute budget. Section 4.4 varies the vendor on a fixed topology. Symptom reports use the plain level of Section 3.4.

Models and harness. We evaluate seven open-weight models, Qwen3.5-397B (Qwen Team, 2026), GLM-4.7 (Zeng et al., 2025), DeepSeek-V4F and DeepSeek-V4-Pro (Xu et al., 2026), GPT-OSS-120B (Agarwal et al., 2025), Qwen3-30B (Yang et al., 2025), and GLM-5.3 (Zeng et al., 2026). NetMend exposes the two tools of Section 3.3 through six harnesses, a native ReAct-style loop (Yao et al., 2023) and five external agent frameworks that run out of process and reach the network through the same tool bridge: OpenClaw, NanoBot, and Hermes, which accept any model, and OpenAI Codex and Claude Code, which run their vendors' closed models (Section 4.3). The main evaluation uses the native harness, which at each step sends the full message history together with the tool schemas, executes every tool call the model returns, and appends each result as a tool message. Decoding uses temperature 0 with at most 16k output tokens per step. Each episode is limited to 250 steps, 600 tool calls, and 1800 s of wall-clock time, identical for every model and harness, and a call issued after the tool budget is exhausted is refused with an instruction to finish. The seven models yield 609 episodes, none of which was excluded.

Metrics. We report Pass@1, the fraction of episodes whose verification targets pass after the agent finishes, measured on one greedy sample per task, the $n = 1$ case of the estimator of Chen et al. (2021), and RCA, the fraction of episodes in which the agent declares or repairs on a node in the

fault's equivalence class (Section 3.3). No-damage is measured on repaired episodes only, since an unrepaired network cannot be tested for collateral damage, and is reported together with the number of repaired episodes. Efficiency is the median ratio of tool calls to the reference solution over solved episodes of the six classes that carry a reference (`physical_link`, `blackhole_route`, `mtu_mismatch`, `netem_loss`, `acl_misconfig`, and `route_policy_misconfig`), whose reference length on the evaluation networks is 5 tool calls, or 6 for `blackhole_route`.

Table 2: Results of NetMend. Best value per column in bold within the main evaluation and within the harness block.

		Correctness					Safety	Efficiency			
Setting	**Model**	**Pass@1**	**Routing**	**Interface**	**Side path**	**RCA**	**No-Dmg.**	**Ratio**	**Calls**	**Time (s)**	**Bud.**
Main evaluation: 29 classes × 3 networks, native harness, 600 tool calls											
	Qwen3.5-397B	87.4	95.8	83.3	84.8	90.8	96.1	3.2	22.6	300	1
	GLM-4.7	87.4	**100.0**	**86.7**	78.8	**98.9**	94.7	4.1	59.2	383	7
	DeepSeek-V4F	**94.3**	95.8	**86.7**	**100.0**	94.3	92.7	**3.1**	**22.3**	476	9
All classes	GPT-OSS-120B	66.7	79.2	63.3	60.6	82.8	91.4	4.2	32.2	**293**	2
	Qwen3-30B	27.6	41.7	30.0	15.2	63.2	87.5	4.9	96.3	383	27
	GLM-5.3	92.0	95.8	83.3	97.0	87.4	**98.8**	3.3	26.9	654	13
	DeepSeek-V4-Pro	85.1	91.7	73.3	90.9	81.6	94.6	3.2	81.8	678	22
Harness: 9 classes × 3 networks, 600 tool calls											
Native	GLM-4.7	**96.3**	**100.0**	**100.0**	88.9	**100.0**	96.2	3.2	48.6	348	1
OpenClaw	GLM-4.7	88.9	**100.0**	77.8	88.9	96.3	**100.0**	4.0	41.0	474	0
NanoBot	GLM-4.7	63.0	77.8	55.6	55.6	70.4	**100.0**	4.6	61.4	752	0
Hermes	GLM-4.7	88.9	88.9	77.8	**100.0**	88.9	**100.0**	4.8	31.9	479	0
Codex	GPT-6-Luna	92.6	**100.0**	77.8	77.8	77.8	**100.0**	3.2	25.6	178	0
Claude Code	Claude Opus 5.5	**96.3**	**100.0**	88.9	**100.0**	96.3	**100.0**	**3.0**	**20.4**	**103**	0
Tool budget: same tasks, GLM-4.7, native harness											
200 tool calls		88.9	100.0	77.8	88.9	92.6	100.0	7.0	46.6	403	2
60 tool calls		88.9	88.9	77.8	100.0	88.9	100.0	4.8	36.6	285	0
Network size: 6 classes × 2 networks per tier (mean node count in parentheses)											
S (27)	GLM-4.7	100.0	100.0	100.0	100.0	100.0	100.0	9.1	58.0	250	0
	Qwen3.5-397B	66.7	100.0	75.0	25.0	75.0	100.0	3.2	18.4	899	4
M (50)	GLM-4.7	100.0	100.0	100.0	100.0	100.0	100.0	4.7	31.1	299	0
	Qwen3.5-397B	83.3	100.0	100.0	50.0	83.3	100.0	2.4	14.5	315	0
L (91)	GLM-4.7	91.7	100.0	100.0	75.0	91.7	100.0	5.1	39.2	604	0
	Qwen3.5-397B	75.0	100.0	100.0	25.0	83.3	100.0	2.9	18.4	262	0
XL (178)	GLM-4.7	100.0	100.0	100.0	100.0	100.0	100.0	6.8	53.6	248	0
	Qwen3.5-397B	91.7	100.0	100.0	75.0	91.7	100.0	3.3	29.0	634	2
Segment, single vendor: 5 link-layer classes × 2 networks, FRR in every segment											
Campus	GLM-4.7	90.0	–	100.0	50.0	90.0	100.0	3.1	18.9	131	0
	GPT-OSS-120B	70.0	–	87.5	0.0	90.0	71.4	5.2	26.9	108	0
	Qwen3.5-397B	80.0	–	100.0	0.0	80.0	100.0	2.8	13.5	274	0
Spine–leaf	GLM-4.7	100.0	–	100.0	100.0	100.0	100.0	3.0	30.0	216	0
	GPT-OSS-120B	100.0	–	100.0	100.0	100.0	100.0	3.6	30.3	99	0
	Qwen3.5-397B	100.0	–	100.0	100.0	100.0	100.0	2.4	14.7	317	0
Hub–spoke	GLM-4.7	80.0	–	100.0	0.0	80.0	100.0	3.5	36.0	597	0
	GPT-OSS-120B	60.0	–	75.0	0.0	80.0	100.0	6.5	35.0	152	0
	Qwen3.5-397B	80.0	–	100.0	0.0	80.0	100.0	3.6	14.6	296	0
Segment, multi-vendor: same tasks, vendor per segment in parentheses											
Campus (FRR)	GLM-4.7	100.0	–	100.0	100.0	100.0	100.0	1.9	15.6	251	0
	GPT-OSS-120B	60.0	–	75.0	0.0	80.0	66.7	3.6	25.8	121	0
	Qwen3.5-397B	80.0	–	100.0	0.0	80.0	87.5	3.2	12.4	117	0
Spine–leaf (VyOS)	GLM-4.7	100.0	–	100.0	100.0	100.0	100.0	17.1	71.3	986	1
	GPT-OSS-120B	80.0	–	75.0	100.0	100.0	100.0	8.9	43.9	254	0
	Qwen3.5-397B	100.0	–	100.0	100.0	100.0	100.0	6.3	34.6	631	0
Hub–spoke (SR Linux)	GLM-4.7	80.0	–	100.0	0.0	80.0	100.0	1.8	42.2	602	2
	GPT-OSS-120B	60.0	–	75.0	0.0	80.0	100.0	4.6	31.1	128	0
	Qwen3.5-397B	80.0	–	100.0	0.0	80.0	100.0	3.1	16.9	216	0

4.2 Main Results

Table 2 reports the main evaluation together with the harness, budget, network-size, and vendor arms of Sections 4.3 and 4.4, and Figure 2 in the appendix breaks Pass@1 down by fault class. Pass@1 is reported over all episodes of a row and per layer of the fault, No-Dmg. over repaired episodes, and Ratio as the median tool-call ratio to the reference solution over the repaired episodes of the classes that carry one. Calls and Time are per-episode means, and Bud. counts the episodes that ended on a budget limit.

Repair success varies widely across models and fault classes. Pass@1 ranges from 27.6% to 94.3%. Four classes are repaired in every episode (`physical_link`, `netem_delay`, `prefix_list_missing`, and `route_policy_misconfig`), while `l2vpn_peer_missing`, `vpn_vrf_missing`, and `ospf_auth_mismatch` are repaired in fewer than 40% of their episodes (Figure 2; Table 6 in the appendix adds RCA and No-Dmg. per class). Grouping classes by the layer at which the fault sits, Pass@1 is 85.7% for routing faults against 72.4% for interface

faults and 75.3% for faults on a side path that the symptom does not point at. Fault class drives this spread.

Localization outpaces repair. Only 10 of the 470 repaired episodes miss the root cause, and RCA exceeds Pass@1 for five of the seven models, so the gap between the two counts episodes in which the agent reached the faulty device but did not complete the repair. Overall, 61 of the 521 correctly localized faults (11.7%) are still not repaired. The gap is largest for Qwen3-30B, which localizes 63.2% of faults but repairs 27.6%, and for GPT-OSS-120B, which localizes 82.8% but repairs 66.7%. GPT-OSS-120B issues a finish declaration in 8 episodes, so its RCA derives almost entirely from repair actions on the faulty device. A report-only evaluation would credit it with a root-cause rate of 9%, whereas the network-state verdict credits it with a repair rate of 66.7%. Root-cause identification and repair are therefore distinct abilities, and NetMend reports both. A three-arm comparison on nine classes (Appendix F) confirms this reading: asked only to diagnose, GPT-OSS-120B answers in two of nine episodes, and diagnosing before repairing raises Pass@1 for neither model.

Collateral damage is rare overall but concentrated. Between 87.5% and 98.8% of repaired episodes leave the rest of the network reachable, and the exceptions cluster in classes whose repair touches route filtering. For `prefix_list_missing` only 23.8% of repairs are clean, and for `route_policy_misconfig` 81.0%. The injected filter admits only a few prefixes, so a repair that re-adds the reported prefix alone restores the reported destination while the other filtered destinations stay unreachable, and only a re-probe of the pairs that traverse the repaired device observes this. Repairs that instead remove the filter restore the baseline and pass the check. The verifier's path-quality check flags no hop-count increase in any repaired episode (Appendix D).

Successful repairs cost several times the reference solution. Median tool-call ratios against the reference solution range from 3.1 (DeepSeek-V4F) to 4.9 (Qwen3-30B), and the mean for Qwen3-30B reaches 9.0. Budget limits matter for two models. Qwen3-30B ends 27 episodes on the step or wall-clock cap and DeepSeek-V4-Pro 22, against at most 13 for any other model. GPT-OSS-120B ends 79 of its episodes without calling the finish action.

Variance across network instances is small relative to differences between models. Across the three generated networks per class, per-model Pass@1 varies with a standard deviation of 1.6 to 4.3 points (largest range 10.3 points for Qwen3.5-397B), and bootstrap 95% intervals over fault classes span 10 to 28 points. Under this bootstrap, 13 of the 21 model pairs are separable, the five models above 85% largely are not, and any two models disagree on 7% to 67% of individual tasks (Figure 4 in the appendix).

4.3 Effect of Harness and Budget

Holding GLM-4.7 and a nine-class subset on three network instances fixed, we vary one axis at a time (Table 2). The harness arm runs the same 27 tasks under the native loop and under OpenClaw, NanoBot, and Hermes at the common budget of 250 steps, 600 tool calls, and 1800 s (108 episodes), plus GPT-6-Luna under Codex and Claude Opus 5.5 under Claude Code on the same tasks (54 episodes). The budget arm keeps the native loop and lowers the tool-call cap from 600 to 200 and to 60 (54 episodes).

The harness moves Pass@1 by more than 30 points on the same weights. The native loop repairs 96.3% of the tasks, OpenClaw and Hermes 88.9%, and NanoBot 63.0%. Hermes uses the fewest tool calls (31.9 per episode against 48.6 for the native loop), whereas NanoBot uses the most (61.4) and takes twice as long (752 s against 348 s). A benchmark result for a model is therefore a result for a model–harness pair, as observed for coding agents (Yang et al., 2024), and NetMend records both.

Closed models under their own harness lead on the hard subset. Claude Opus 5.5 under Claude Code repairs 96.3% of the 27 tasks with 20.4 tool calls and 103 s per episode, and GPT-6-Luna under Codex 92.6% with 25.6 calls and 178 s, against 48.6 calls and 348 s for GLM-4.7 under the native loop at 96.3%. On the 18-task hard subset of Appendix E they reach 94.4% and 88.9% against 72.2% for the best open-weight model, so the frontier nearly saturates even this subset and headroom lies in harder classes and compound tasks; model and harness change together in these pairs, so they rank systems rather than harnesses.

A tighter tool budget changes the trajectory more than the outcome. Lowering the cap from 600 to 200 and to 60 calls moves Pass@1 from 96.3% to 88.9% at both lower caps, a difference of one task, while mean tool calls fall from 48.6 to 46.6 and 36.6 and wall-clock time from 348 s to 285 s at the 60-call cap.

4.4 Effect of Network Scale and Vendor

Network size does not drive difficulty between 27 and 178 nodes. The generator produces four size tiers, S, M, L, and XL, with mean sizes of 27, 50, 91, and 178 nodes and two instances per tier. Six fault classes are paired by scope, so that each layer contributes one fault whose evidence spreads along a path and one that sits on a single device, and GLM-4.7 and Qwen3.5-397B are run on every task (96 episodes, Table 2). GLM-4.7 passes 100%, 100%, 91.7%, and 100% from S to XL, and Qwen3.5-397B 66.7%, 83.3%, 75.0%, and 91.7%, with no monotone trend for either model. Mean tool calls per episode do not grow with size for either model, routing-layer faults are repaired in every tier by both, and the variation between tiers comes from the interface and side-path faults. The agent's search does not expand with the network; the symptom names the probe endpoints and the documentation gives the topology, so the candidate set is bounded before the search starts.

For link-layer faults, the segment matters more than the vendor. Three blueprints, a three-tier campus, a spine–leaf fabric, and a hub–spoke network, are chained into one composite topology, and the same five link-layer fault classes (`mtu_mismatch`, `netem_loss`, `netem_delay`, `bandwidth_saturated`, and `arp_static_wrong`) are injected into each segment on two instances. The vendor comparison is restricted to link-layer classes because routing-layer faults cannot be injected identically on all three systems. In the control arm every segment runs FRR, and GLM-4.7, Qwen3.5-397B, and GPT-OSS-120B yield 90 episodes. All three models pass 100% in the spine–leaf segment, against 90.0%, 80.0%, and 70.0% in the campus segment and 80.0%, 80.0%, and 60.0% in the hub–spoke segment (Table 2), so the position of a fault inside one topology moves the rate by 20 to 40 points under a single vendor. In the multi-vendor arm the three segments run FRR, VyOS, and SR Linux, and the same models are run on the same 90 tasks. The outcome is identical to the control arm on all but eight tasks, of which seven belong to GPT-OSS-120B, five repairs lost and two gained, and one to GLM-4.7, a repair gained. For Qwen3.5-397B the vendor effect on link-layer faults is zero, and the difference between segments is a property of the topology; whether routing-layer faults, whose repair commands differ across systems, show a vendor effect is untested.

Validity checks. Appendices E and G report four checks of the design: per-class Pass@1 on generated networks correlates with production-derived networks (Spearman ρ of 0.787, per-class means within one point), no passing trajectory is shorter than the reference solution of its task, compound tasks differ from the conjunction of their components in 8 of 30 episodes, and unaudited symptom wording raises Pass@1 by 7 points, so all reported results use the audited level.

5 Conclusion

We presented NetMend, a benchmark that evaluates LLM agents on the repair of injected faults in live containerized networks. A generator produces heterogeneous topologies across vendors and protocols, every task is a replayable specification, and every verdict is read from the network state. Across seven open-weight models, repair success spans 28% to 94%; about one localized fault in nine remains unrepaired, and in the classes whose fix touches route filtering, repairs that pass the target check still break unrelated reachability. Neither effect registers in diagnosis-level or static-artifact evaluation, and both decide whether an agent can be trusted with an operational network. The closed systems under their vendors' harnesses nearly saturate even the hard subset, so headroom at the frontier lies in harder classes and higher-order compound tasks, which enter the bank as specifications without changes to the harness; production traffic, free-form command execution, and change-control constraints remain outside the present scope. Our next step is the harness: since the same weights move by more than 30 points across frameworks, a network-specific repair loop with probe-guided localization and reachability checks around every change is the lever we will study next.

AI USE STATEMENT

In this work, we used generative AI tools to assist with editing and polishing the text, with figure layout, and with writing analysis and plotting scripts. We have not used generative AI tools to produce experimental data or results. The LLMs that appear in the experiments are the subjects under evaluation, driven by the released harness. All AI-assisted code was reviewed by the authors, and every number reported in the paper was checked against the raw experiment logs. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pp. 419–435, 2018.

Rufat Asadli, Benjamin Hoffman, Ioannis Protogeros, and Laurent Vanbever. Evaluating agentic configuration repair for computer networks. In *ICML Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD)*, 2026. arXiv:2606.06212.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016.

Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 155–168, 2017.

Yuanhong Cai, Xiaohui Nie, Kanglin Yin, Changhua Pei, Yongqian Sun, Shenglin Zhang, Haibin Liu, Guiyang Liu, Xidao Wen, Fang Situ, et al. A multi-dataset benchmark for evaluating LLM agents in microservice failure diagnosis. *arXiv preprint arXiv:2606.29193*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 674–688, 2024.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. In *Proceedings of Machine Learning and Systems*, 2025.

Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 469–483, 2015.

FRRouting Project. FRRouting: The FRR protocol suite, 2024. `https://frrouting.org`.

Jiaqi Gao, Nofel Yaseen, Robert MacDavid, Felipe Vieira Frujeri, Vincent Liu, Ricardo Bianchini, Ramaswamy Aditya, Xiaohang Wang, Henry Lee, David Maltz, Minlan Yu, and Behnaz Arzani. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 253–269, 2020.

Vipul Harsh, Tong Meng, Kapil Agrawal, and Philip Brighten Godfrey. Flock: Accurate network fault localization at scale. *Proc. ACM Netw.*, 1(CoNEXT1), 2023a.

Vipul Harsh, Wenxuan Zhou, Sachin Ashok, Radhika Niranjan Mysore, Brighten Godfrey, and Sujata Banerjee. Murphy: Performance diagnosis of distributed cloud applications. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pp. 438–451, 2023b.

Saurabh Jha, Rohan R. Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, Noah Zheutlin, Saki Takano, Divya Pathak, Felix George, Xinbo Wu, Bekir O Turkkan, Gerard Vanloo, Michael Nidd, Ting Dai, Oishik Chatterjee, Pranjal Gupta, Suranjana Samanta, Pooja Aggarwal, Rong Lee, Jae wook Ahn, Debanjana Kar, Amit Paradkar, Yu Deng, Pratibha Moogi, Prateeti Mohapatra, Naoki Abe, Chandrasekhar Narayanaswami, Tianyin Xu, Lav R. Varshney, Ruchi Mahindru, Anca Sailer, Laura Shwartz, Daby Sow, Nicholas C. M. Fuller, and Ruchir Puri. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In *Forty-second International Conference on Machine Learning*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, pp. 54107–54157, 2024.

Sebastien Levy, Randolph Yao, Youjiang Wu, Yingnong Dang, Peng Huang, Zheng Mu, Pu Zhao, Tarun Ramani, Naga Govindaraju, Xukun Li, Qingwei Lin, Gil Lapid Shafriri, and Murali Chintalapati. Predictive and adaptive failure mitigation to avert production cloud VM interruptions. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 1155–1170, 2020.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, pp. 52989–53046, 2024.

Yuhe Liu, Changhua Pei, Longlong Xu, Bohan Chen, Mingze Sun, Zhirui Zhang, Yongqian Sun, Shenglin Zhang, Kun Wang, Haiming Zhang, Jianhui Li, Gaogang Xie, Xidao Wen, Xiaohui Nie, Minghua Ma, and Dan Pei. OpsEval: A comprehensive benchmark suite for evaluating large language models' capability in IT operations domain. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pp. 503–513, 2025.

Ali Maatouk, Fadhel Ayed, Nicola Piovesan, Antonio De Domenico, Merouane Debbah, and Zhi-Quan Luo. TeleQnA: A benchmark dataset to assess large language models telecommunications knowledge. *IEEE Network*, 40(2):253–260, 2026.

Sathiya Kumaran Mani, Yajie Zhou, Kevin Hsieh, Santiago Segarra, Trevor Eberl, Eliran Azulai, Ido Frizler, Ranveer Chandra, and Srikanth Kandula. Enhancing network management using code generated by large language models. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pp. 196–204, 2023.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, pp. 9025–9049, 2024.

Rajdeep Mondal, Alan Tang, Ryan Beckett, Todd Millstein, and George Varghese. What do LLMs need to synthesize correct router configurations? In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pp. 189–195, 2023.

Pooria Namyar, Arvin Ghavidel, Daniel Crankshaw, Daniel S. Berger, Kevin Hsieh, Srikanth Kandula, Ramesh Govindan, and Behnaz Arzani. Enhancing network failure mitigation with Performance-Aware ranking. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pp. 335–357, 2025.

Nokia and contributors. Containerlab: Container-based networking labs, 2024. `https://containerlab.dev`.

Ioannis Protogeros, Rufat Asadli, Benjamin Hoffman, and Laurent Vanbever. Benchmarking LLM-driven network configuration repair. *arXiv preprint arXiv:2604.22513*, 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

Kuan-Hao Tseng, Niruth Bogahawatta, Yasod Ginige, Kosta Dakic, Arunan Sivanathan, and Suranga Seneviratne. SADE: Symptom-aware diagnostic escalation for LLM-based network troubleshooting. In *2026 IEEE 51st Conference on Local Computer Networks (LCN)*, pp. 1–12, 2026.

Ahmed Twabi, Yepeng Ding, and Tohru Kondo. NetAgentBench: A state-centric benchmark for evaluating agentic network configuration. *arXiv preprint arXiv:2604.09678*, 2026.

Changjie Wang, Mariano Scazzariello, Alireza Farshin, Simone Ferlin, Dejan Kostić, and Marco Chiesa. NetConfEval: Can LLMs facilitate network configuration? *Proceedings of the ACM on Networking*, 2(CoNEXT2), 2024a.

Chenxu Wang, Xumiao Zhang, Runwei Lu, Xianshang Lin, Xuan Zeng, Xinlei Zhang, Zhe An, Gongwei Wu, Jiaqi Gao, Chen Tian, Guihai Chen, Guyue Liu, Yuhong Liao, Tao Lin, Dennis Cai, and Ennan Zhai. Towards LLM-based failure localization in production-scale networks. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pp. 496–511, 2025a.

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

Zhaodong Wang, Samuel Lin, Guanqing Yan, Soudeh Ghorbani, Minlan Yu, Jiawei Zhou, Nathan Hu, Lopa Baruah, Sam Peters, Srikanth Kamath, Jerry Yang, and Ying Zhang. Intent-driven network management with multi-agent LLMs: The Confucius framework. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pp. 347–362, 2025b.

Zhihao Wang, Alessandro Cornacchia, Franco Galante, Carlo Centofanti, Alessio Sacco, and Dingde Jiang. Towards a playground to democratize experimentation and benchmarking of AI agents for network troubleshooting. In *Proceedings of the 1st Workshop on Next-Generation Network Observability*, 2025c.

Zhihao Wang, Alessandro Cornacchia, Alessio Sacco, Franco Galante, Marco Canini, and Dingde Jiang. A network arena for benchmarking AI agents on network troubleshooting. *arXiv preprint arXiv:2512.16381*, 2025d.

Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. NetLLM: Adapting large language models for networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 661–678, 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, pp. 52040–52094, 2024.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, pp. 50528–50652, 2024.

Yitao Yang and Hong Xu. NetOpsBench: Open arena for NetOps in AI infrastructure, 2026. URL `https://github.com/netx-lab/NetOpsBench`.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. GLM-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Lingzhe Zhang, Yunpeng Zhai, Tong Jia, Chiming Duan, Minghua He, Leyi Pan, Zhaoyang Liu, Bolin Ding, and Ying Li. MicroRemed: Benchmarking LLMs in microservices remediation. *arXiv preprint arXiv:2511.01166*, 2025.

Mengyu Zheng, Kai Han, Boxun Li, Haiyang Xu, Yuchuan Tian, Wei He, Hang Zhou, Jianyuan Guo, Hailin Hu, Lin Ma, et al. Claw-SWE-Bench: A benchmark for evaluating OpenClaw-style agent harnesses on coding tasks. *arXiv preprint arXiv:2606.12344*, 2026.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, pp. 15585–15606, 2024.

Yajie (Lesley) Zhou, Jiajun Ruan, Eric Wang, Sadjad Fouladi, Francis Yan, Kevin Hsieh, and Zaoxing Liu. NetArena: Dynamic benchmarks for AI agents in network automation. In *International Conference on Learning Representations*, pp. 121053–121081, 2026.

A Comparison with Existing Environments

Table 3 extends Table 1 to the environment and task-generation dimensions. Protocols counts the distinct routing and forwarding protocols a benchmark runs, and NOS counts the network operating systems it supports within one benchmark. Replay marks benchmarks whose tasks store a recipe that rebuilds the network and re-injects the fault, Compound marks tasks with several simultaneous root causes, and Prod.-Topo marks tasks rebuilt from production vendor configurations. Values not reported in a paper are taken from its open-source repository where available. NetMend is the only environment that combines a running network operating system with compound faults and production-derived topologies.

Table 3: Environment and task-generation comparison.

Benchmark	Protocols	NOS	Replay	Compound	Prod.-Topo
NIKA	3	2	✓	×	×
NetOpsBench	2	1	✓	×	×
NetArena	0	0	✓	×	×
Cornetto	4	—	✓	✓	×
NetMend (ours)	8	3	✓	✓	✓

B FAULT CLASSES

Table 5 lists the 29 single-fault classes. Each class is defined by the device-level change the injector applies, so the same class instantiates on any network that hosts the corresponding protocol or feature.

Action space. The repair tool exposes 23 typed write actions, listed in Table 4 with the classes they restore. The space is closed: the agent chooses an action and its arguments, and the tool renders the operator command, so the skill under test is choosing the right change on the right device and interface rather than composing vendor syntax. Six actions serve more than one class, and the arguments distinguish the cases, for example the interface and value of `set_mtu` or the prefix and next hop of `route_set`. In the main evaluation every one of the 23 actions appears in at least one passing episode, and for eight classes every passing episode used one and the same action; the remaining classes were also repaired by other actions on other devices, which the verifier accepts only when the network state meets the targets. A free-form command interface, in which the agent writes the vendor command itself, is not part of this release.

Table 4: The 23 write actions of `net_repair` and the fault classes they restore.

Action	Restores
`if_up`	`physical_link`
`route_set`	`static_route_missing; with route_del, static_route_wrong`
`route_del`	`static_route_wrong, l3_loop`
`route_del_blackhole`	`blackhole_route`
`set_redistribute`	`route_policy_misconfig`
`set_mtu`	`mtu_mismatch`
`clear_netem`	`netem_loss, netem_delay, bandwidth_saturated`
`fw_del`	`acl_misconfig`
`set_ip`	`interface_ip_error, loopback_conflict`
`arp_del`	`arp_static_wrong`
`prefix_list_add`	`prefix_list_missing`
`ospf_auth_clear`	`ospf_auth_mismatch`
`bgp_remote_as`	`bgp_neighbor_misconfig`
`isis_iface_enable`	`isis_misconfig`
`fdb_del`	`mac_table_wrong`
`nat_masquerade_set`	`nat_rule_missing, nat_wrong_interface`
`vrrp_enable`	`ha_protocol_disabled`
`stp_enable`	`stp_global_disabled, stp_port_disabled`
`vlan_set_pvid`	`vlan_misconfig`
`ldp_iface_enable`	`l3vpn_ldp_broken`
`vrf_bind`	`vpn_vrf_missing`
`vxlan_rebuild`	`vxlan_misconfig`
`vxlan_peer_add`	`l2vpn_peer_missing`

The difficulty of a task is computed from its fault class. Five ordinal factors $x_j(\tau) \in \{1, 2, 3\}$, $j \in \mathcal{J} = \{\mathrm{L}, \mathrm{O}, \mathrm{D}, \mathrm{R}, \mathrm{S}\}$, encode the network layer, the observability, the diagnostic complexity, the recovery complexity, and the impact scope. The observability factor follows the verification modality of the class, with probes, device state, and routing tables at 1, 2, and 3. The difficulty of a single-fault task is the weighted sum of the factors, and the difficulty of a compound task takes each factor's maximum over the components and scales the sum by the number of components:

$$d(\tau) = \sum_{j \in \mathcal{J}} w_j \, x_j(\tau), \qquad d(\tau_1 \oplus \cdots \oplus \tau_k) = \Big(\sum_{j \in \mathcal{J}} w_j \max_i x_j(\tau_i) \Big) \big(1 + 0.1\,(k-1)\big), \tag{5}$$

where $w = (0.35, 0.15, 0.30, 0.10, 0.10)$ in the order of $\mathcal{J}$. Scores below 1.9 form the easy band, scores in $[1.9, 2.4)$ the medium band, and scores of 2.4 and above the hard band. The weights and the band thresholds are fixed constants of the release. The difficulty column of Table 5 follows Equation 5 and ranges from 1.35 to 2.75. Routing classes cover unicast routing protocols, static routes, and MPLS L3VPN, switching classes cover layer-2 forwarding, redundancy protocols, and VXLAN, connectivity classes cover physical and link-parameter faults, and policy classes cover filtering, route policy, and NAT.

Table 5: The 29 single-fault classes. Category is the grouping used in the text, Injected change is the device-level command the injector applies, and Diff. is the computed difficulty.

Fault class	Category	Injected change	Diff.
`blackhole_route`	Routing	null route to the victim's loopback on its uplink router	1.40
`bgp_neighbor_misconfig`	Routing	wrong remote AS on one side of a BGP session	1.65
`ospf_auth_mismatch`	Routing	MD5 authentication enabled on one side of an OSPF link	1.90
`loopback_conflict`	Routing	a core router's loopback address duplicated on an access router	1.95
`static_route_wrong`	Routing	static route to the victim with an unresolvable next hop	2.00
`l3_loop`	Routing	static route on a distribution router that points back toward the core	2.00
`isis_misconfig`	Routing	IS–IS removed from one interface of an adjacency	2.20
`static_route_missing`	Routing	static route to the victim deleted	2.20
`l3vpn_ldp_broken`	Routing	LDP removed from one MPLS interface of a PE router	2.30
`vpn_vrf_missing`	Routing	customer interface detached from its VRF	2.40
`arp_static_wrong`	Switching	permanent ARP entry with a wrong MAC for a link peer	2.45
`ha_protocol_disabled`	Switching	VRRP removed on both gateways of a segment	2.45
`mac_table_wrong`	Switching	static FDB entry that pins a host's MAC to the wrong port	2.45
`l2vpn_peer_missing`	Switching	remote VTEP entry and its learned MACs deleted	2.65
`stp_global_disabled`	Switching	spanning tree disabled on a bridge with redundant trunks	2.65
`vlan_misconfig`	Switching	host access port moved to the wrong VLAN	2.65
`stp_port_disabled`	Switching	one trunk port forced to forwarding with STP off	2.75
`vxlan_misconfig`	Switching	VXLAN interface recreated with a wrong VNI	2.75
`physical_link`	Connectivity	access uplink interface brought down	1.35
`netem_loss`	Connectivity	30% packet loss on an access uplink	1.80
`netem_delay`	Connectivity	200 ms delay on an access uplink	1.80
`bandwidth_saturated`	Connectivity	uplink rate limited to 64 kbit/s	2.10
`interface_ip_error`	Connectivity	interface readdressed outside every planned subnet	2.20
`mtu_mismatch`	Connectivity	uplink MTU lowered below the jumbo baseline	2.20
`acl_misconfig`	Policy	forwarding rule that drops traffic to the victim's loopback	2.00
`prefix_list_missing`	Policy	prefix-list entry removed at a protocol boundary	2.10
`route_policy_misconfig`	Policy	route map at a protocol boundary filters the victim's prefix	2.20
`nat_rule_missing`	Policy	masquerade rule removed on the NAT gateway	2.35
`nat_wrong_interface`	Policy	masquerade rule moved to an inside interface	2.75

C Per-Class Results

Figure 2 shows Pass@1 per fault class and model, and Table 6 complements it with RCA and No-Dmg. per fault class, pooled over the seven models. RCA is at least as high as Pass@1 in every class but `interface_ip_error`, and the gap is largest for `nat_wrong_interface`, where the agent reaches the faulty device in every episode but completes the repair in 61.9%. Collateral damage is confined to seven classes, and the two route-filtering classes, `prefix_list_missing` and `route_policy_misconfig`, account for most of it. The three classes below 40% Pass@1 are `vpn_vrf_missing` (28.6%), `ospf_auth_mismatch` (33.3%), and `l2vpn_peer_missing` (38.1%), followed by `interface_ip_error`, `l3vpn_ldp_broken`, and `loopback_conflict` at 57.1%.

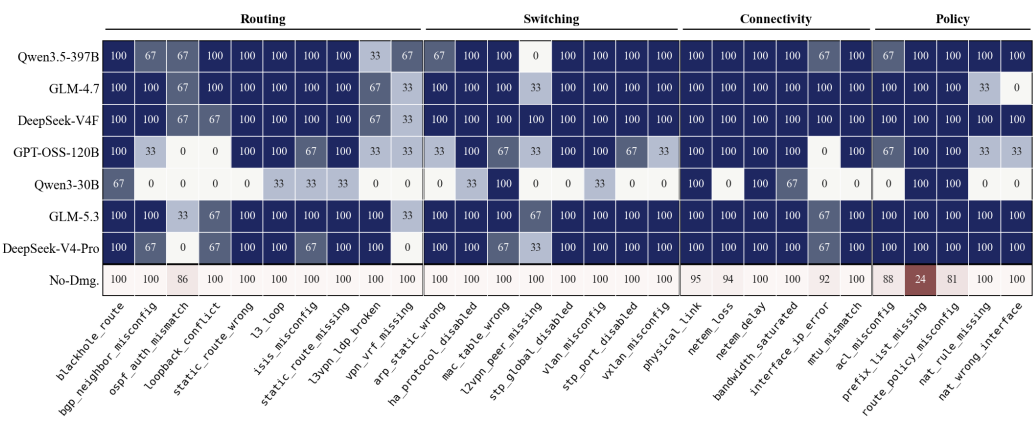


Figure 2: Pass@1 per fault class and model, with the share of clean repairs per class (No-Dmg.) in the last row.

Table 6: Per-class results of the main evaluation. Each class aggregates 21 episodes (7 models $\times$ 3 seeds). Pass@1 and RCA are percentages of episodes, and No-Dmg. the percentage of repaired episodes without collateral damage.

Fault class	Category	Pass@1	RCA	No-Dmg.
blackhole_route	Routing	95.2	95.2	100.0
bgp_neighbor_misconfig	Routing	66.7	66.7	100.0
ospf_auth_mismatch	Routing	33.3	47.6	85.7
loopback_conflict	Routing	57.1	57.1	100.0
static_route_wrong	Routing	85.7	100.0	100.0
l3_loop	Routing	90.5	95.2	100.0
isis_misconfig	Routing	81.0	85.7	100.0
static_route_missing	Routing	90.5	100.0	100.0
l3vpn_ldp_broken	Routing	57.1	76.2	100.0
vpn_vrf_missing	Routing	28.6	42.9	100.0
arp_static_wrong	Switching	71.4	71.4	100.0
ha_protocol_disabled	Switching	90.5	100.0	100.0
mac_table_wrong	Switching	90.5	95.2	100.0
l2vpn_peer_missing	Switching	38.1	61.9	100.0
stp_global_disabled	Switching	85.7	95.2	100.0
vlan_misconfig	Switching	90.5	100.0	100.0
stp_port_disabled	Switching	81.0	81.0	100.0
vxlan_misconfig	Switching	76.2	100.0	100.0
physical_link	Connectivity	100.0	100.0	95.2
netem_loss	Connectivity	85.7	95.2	94.4
netem_delay	Connectivity	100.0	100.0	100.0
bandwidth_saturated	Connectivity	95.2	95.2	100.0
interface_ip_error	Connectivity	57.1	42.9	91.7
mtu_mismatch	Connectivity	85.7	85.7	100.0
acl_misconfig	Policy	76.2	100.0	87.5
prefix_list_missing	Policy	100.0	100.0	23.8
route_policy_misconfig	Policy	100.0	100.0	81.0
nat_rule_missing	Policy	66.7	90.5	100.0
nat_wrong_interface	Policy	61.9	100.0	100.0

D PATH-QUALITY CHECK

Beside collateral damage, the verifier compares every re-probed pair with its baseline path (Section 3.3). A pair p is a quality regression when $\mathrm{hops}(p) > \mathrm{hops}_0(p)$, or when $\mathrm{rtt}(p) > 3\,\mathrm{rtt}_0(p)$ and $\mathrm{rtt}(p) - \mathrm{rtt}_0(p) > 0.5\,\mathrm{ms}$, where the subscript 0 denotes the baseline value, and the quality credit holds when no pair in P_Δ regresses. Figure 3 summarizes this check over the 470 repaired episodes of the main evaluation. Twenty-two episodes carry at least one flag, 34 pairs in total, and none of them is a hop-count increase. Every flag is a round-trip time that rose from about 0.1 ms to between 0.6 and 30 ms on an unchanged path, which we attribute to host load during the campaign rather than to the repair. No repair in the main evaluation therefore lengthened a path, and the round-trip threshold of 0.5 ms is too tight for a containerized testbed and is reported as a limitation of the current release.

Design intent. Four classes raise the question whether a repair that passes the check can still violate the intended design, for example by removing a filter instead of correcting it. In NetMend the injected change of these classes is an addition to the baseline: `acl_misconfig` appends an `iptables` drop rule, `ospf_auth_mismatch` enables authentication on one side of a link, and `prefix_list_missing` and `route_policy_misconfig` attach a filtering route-map to a redistribution that the baseline runs unfiltered. Removing the added rule, key, or filter therefore restores the baseline, and the re-probe of every baseline pair through the modified device is what separates the two repairs of `prefix_list_missing`: re-adding one entry to the injected list passes the target probe but leaves the other prefixes filtered and fails the collateral check, whereas detaching the filter restores every pair. In the main evaluation, 15 of the 21 passing episodes of this class re-added an entry and 16 of the 21 were flagged for collateral damage; all seven passing episodes of `ospf_auth_mismatch` cleared the injected one-sided key, and 18 of the 21 passing episodes of `route_policy_misconfig` re-attached the unfiltered redistribution. The check reads reachability and paths and does not diff configurations, so a change that alters neither is not detected; a configuration-level diff against the baseline is left to future work.

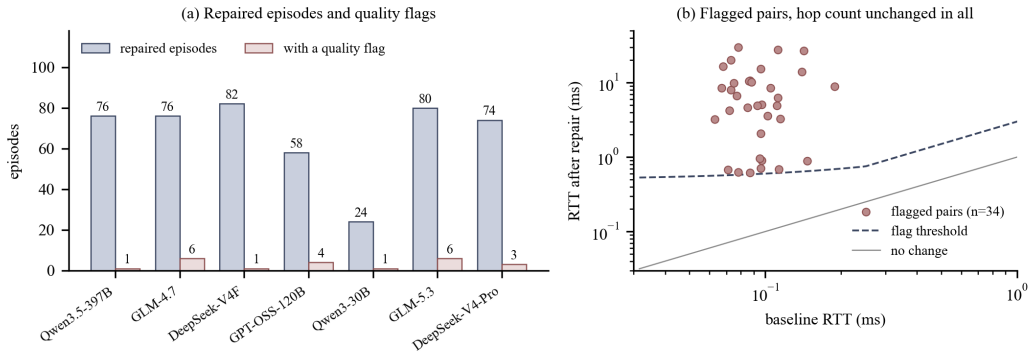


Figure 3: Path-quality check on the main evaluation. (a) Repaired episodes per model and those with a quality flag. (b) The 34 flagged pairs, baseline against post-repair round-trip time, with the flagging rule.

E INSTANCE VARIANCE AND VALIDITY CHECKS

Figure 4 shows, for each model, Pass@1 on each of the three generated networks of the main evaluation, the mean over the 87 episodes, and the bootstrap 95% interval over fault classes. The three networks of a model lie within 3.5 to 10.3 points of each other, whereas the intervals of the five models above 85% overlap and those of the remaining two do not overlap with them.

A hard subset separates the leading models further. The six classes whose pooled Pass@1 falls below 60% (`interface_ip_error`, `l2vpn_peer_missing`, `l3vpn_ldp_broken`, `loopback_conflict`, `ospf_auth_mismatch`, `vpn_vrf_missing`) form a hard subset of 18 tasks. Table 7 reports Pass@1 on this subset with a bootstrap 95% interval over classes: the five open-weight models above 85% overall spread from 44.4% to 72.2% here, GPT-OSS-120B and

Qwen3-30B fall to 16.7% and 0.0%, and the intervals of the leaders still overlap at 18 tasks. The two closed systems of Section 4.3 were run on the same 18 tasks under the same environment. Claude Opus 5.5 under Claude Code passes 17 of 18 and names the root cause in all 18; its one miss is `ospf_auth_mismatch` on one network, and one of its repairs is flagged for collateral damage. GPT-6-Luna under Codex passes 16 of 18, one task fewer, and both closed systems exceed the best open-weight model on this subset by more than 16 points.

Table 7: Pass@1 on the hard subset (six classes, 18 tasks per system) with a bootstrap 95% interval over classes; open-weight models under the native harness, closed models under their vendors' harnesses.

System	Pass@1 (%)	95% interval
Claude Opus 5.5 (Claude Code)	94.4	[83.3, 100.0]
GPT-6-Luna (Codex)	88.9	[77.8, 100.0]
DeepSeek-V4F	72.2	[55.6, 88.9]
GLM-4.7	66.7	[44.4, 88.9]
GLM-5.3	61.1	[44.4, 77.8]
Qwen3.5-397B	55.6	[27.8, 77.8]
DeepSeek-V4-Pro	44.4	[11.1, 72.2]
GPT-OSS-120B	16.7	[5.6, 27.8]
Qwen3-30B	0.0	[0.0, 0.0]

Class-internal variance. Across the three generated networks the per-class Pass@1 pooled over models has a median standard deviation of 6.7 points; 9 classes obtain the same rate on all three networks, and the largest spread is 24.3 points for `ospf_auth_mismatch`, whose adjacency reforms only after the far end's dead interval. Each class is instantiated with a fixed injected value (an MTU of 1000, 30% loss), so the variation within a class comes from the location and the network; randomizing these values is left to a later release.

Vendor and tool cost. The vendor does change the cost: in the VyOS segment GLM-4.7 issues 71 tool calls per episode against 16 in the FRR segment, re-issuing ping, configuration, and ARP views 20 to 26 times per episode while 10 to 25 calls per episode return errors, which is the origin of its ratio of 17.1 in Table 2.

The following checks concern the benchmark design rather than a single model.

Synthetic difficulty transfers to production-derived networks. On the eleven fault classes that both settings support, five models were run on two production-derived topologies (70 episodes). Per-class Pass@1 on generated networks correlates with the production-derived ones at Spearman $\rho = 0.787$, and the per-class means are within one point (80.0% against 80.9%). The two production topologies support different subsets of classes, eleven and three, so rates are averaged per class rather than pooled over episodes. Rankings measured on generated networks carry over to networks rebuilt from real vendor configurations (Figure 5).

Reference solutions are checked from two independent directions. The search-based reference and the template of the known fix are derived independently for every task of the six reference classes (162 problems in the generator's problem bank), and the 114 passing episodes of the seven models on these classes serve as an empirical check: none is shorter than the reference, and the shortest uses 8 calls against a reference of 5.

Compound tasks measure interference, not conjunction. Each compound episode is compared with the conjunction of its components' outcomes in the main evaluation (Appendix G). The two agree in 22 of 30 episodes, and in every disagreement but one both components are repaired in isolation while their composition is not; on this sample of two classes and 30 episodes, a compound task behaves as a different task from the conjunction of its parts.

Unaudited symptoms leak the answer. Rewriting the audited symptom into the casual phrasing an operator might type raises Pass@1 by 7 points and RCA by 11 to 15 points (Appendix G), so all reported results use the audited level.

Figure 5 shows the transfer check at the level of fault classes: the eleven classes that both settings support lie close to the diagonal. The computed difficulty score of Section 3.4 orders the classes only weakly, with every band containing classes from 30% to 100%. Episodes in the easy band

pass 89.5%, against 76.9% and 71.4% for the medium and hard bands, and the per-class Spearman correlation between score and Pass@1 is -0.280. The layer factor alone separates routing faults from the other two layers (85.7% against 72.4% and 75.3%), and its ordering is monotone for three of the seven models. We therefore ship difficulty as reproducible metadata for stratified sampling and do not use it as a performance predictor.

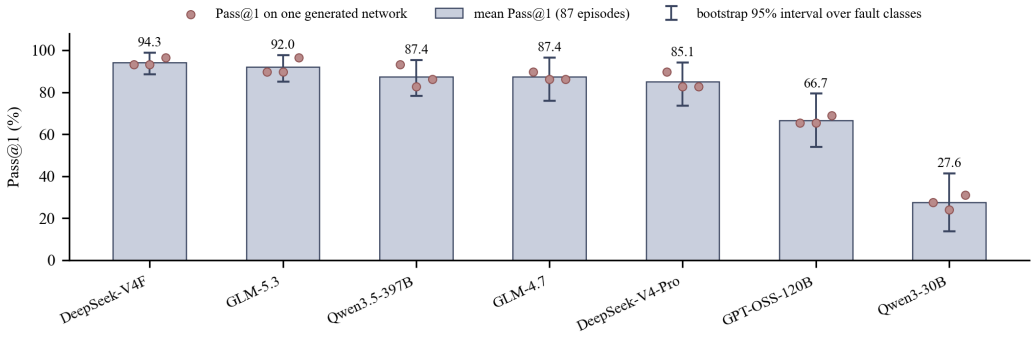


Figure 4: Pass@1 per model on the three generated networks, its mean, and the bootstrap 95% interval over fault classes.

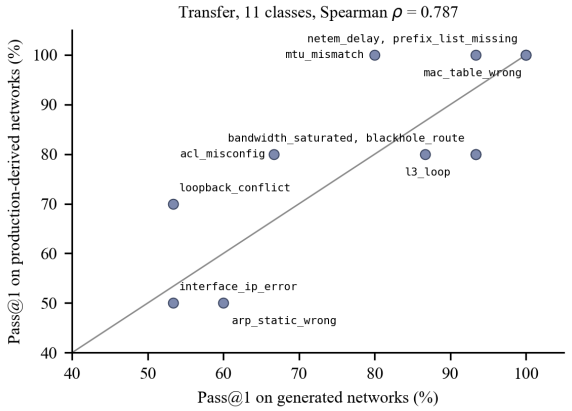


Figure 5: Per-class Pass@1 on generated against production-derived networks for the five models that ran both.

F Diagnosis versus Repair

On a nine-class subset we compare three arms with GLM-4.7 and GPT-OSS-120B, namely diagnosis only, direct repair, and a two-stage pipeline that first diagnoses and then repairs with its own diagnosis in the prompt (Table 9).

Repair evaluation scores ability that a report-only evaluation misses. In the direct-repair arm GLM-4.7 names the root cause in every episode and repairs every task. GPT-OSS-120B repairs 44.4% of the same tasks but never issues a finish declaration, so a report-only evaluation scores it at zero on tasks it has demonstrably fixed. Every one of its repairs touched the root-cause device, which the network-state verdict records and a report does not.

Asked only to diagnose, one model declines to answer. In the diagnosis-only arm GLM-4.7 answers every task and names the correct root cause in 66.7%. GPT-OSS-120B ends seven of nine episodes without stating an answer and is correct both times it does, so its 22.2% is a reporting failure rather than a diagnostic one. A report-only benchmark cannot separate these two cases.

Diagnosing first does not help. Feeding the agent its own diagnosis before repairing leaves GLM-4.7 unchanged at 100% and moves GPT-OSS-120B from 44.4% to 33.3%. At nine episodes per arm the latter gap is one episode wide and is not read as an effect.

Why localized faults stay unrepaired. The 61 episodes of the main evaluation in which the agent reached the root-cause device but the network did not pass fall into four modes, read from the transcripts (Table 8). In 33 the agent acted on the right device with a wrong action, interface, or value, for example `set_ip` on the wrong interface or a redistribution change for an ACL fault; this mode is concentrated in GPT-OSS-120B and Qwen3-30B. In 24 the agent issued at least one repair and then ran out of steps or time, often repeating the same call on the same interface; 17 of these belong to Qwen3-30B. Two episodes repaired only another device, and two ended on an endpoint failure. No localized episode ended with a declaration and no repair attempt, so the gap between RCA and Pass@1 is a gap of execution, not of intent.

Table 8: Failure modes of the localized-but-unrepaired episodes of the main evaluation.

Mode	Episodes	By model
Wrong action, interface, or value on the root-cause device	33	GPT-OSS-120B 14, Qwen3-30B 13, GLM-4.7 3, Qwen3.5-397B 3
Repair issued, then step or time budget exhausted	24	Qwen3-30B 17, GLM-4.7 5, DeepSeek-V4-Pro 1, DeepSeek-V4F 1
Repair on another device only	2	Qwen3-30B 2
Endpoint failure	2	GLM-4.7 2

Table 9: Diagnosis versus repair on nine classes, 9 episodes per cell. Pass@1 is undefined in the diagnosis-only arm.

	GLM-4.7		GPT-OSS-120B	
Arm	**Pass@1**	**RCA**	**Pass@1**	**RCA**
Diagnosis only	–	66.7	–	22.2
Direct repair	100.0	100.0	44.4	66.7
Diagnose, then repair	100.0	100.0	33.3	66.7

G Symptom Wording and Compound Tasks

Figure 6 compares the audited plain symptom of Section 3.4 with a casual rewrite on the same tasks. The rewrite leaks the fault type for 27 of the 29 classes and the faulty node for 16, and it is evaluated on 27 paired tasks per model with GLM-4.7 and Qwen3.5-397B. The leaky wording raises Pass@1 by 7 points and RCA by 11 to 15 points, and the agent needs fewer tool calls because it no longer searches. The gain in RCA exceeds the gain in Pass@1 for both models, so the leak mainly hands the agent the answer to the diagnostic question, and every result in the paper therefore uses the audited level.

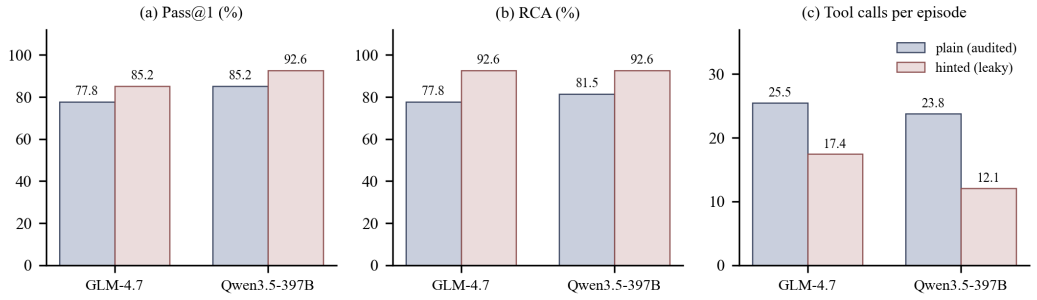


Figure 6: Audited (plain) against leaky (hinted) symptom wording, 27 paired tasks per model.

Table 10 compares each compound episode with the conjunction of its components' outcomes in the main evaluation. The two compound classes are run for five models and three instances (30 episodes). For `blackhole_route` combined with `netem_loss`, the conjunction predicts 12 passes and 9 are observed, and the two agree in 12 of 15 episodes. For `mtu_mismatch` combined with `acl_misconfig`, the conjunction predicts 10 passes and 7 are observed, and the two agree

in 10 of 15 episodes. In every disagreement but one, both components are repaired in isolation but their composition is not, so compound classes add signal that the single-fault classes do not carry.

Table 10: Compound tasks against the conjunction of their components' outcomes in the main evaluation. Conj. counts the episodes in which both components are repaired in isolation, Obs. the episodes in which the compound task is repaired, and Agree the episodes in which the two coincide.

Compound class	Conj.	Obs.	Agree
`blackhole_route` + `netem_loss`	12/15	9/15	12/15
`mtu_mismatch` + `acl_misconfig`	10/15	7/15	10/15

H A Complete Episode

Figures 7, 8, and 9 show one unmodified episode of DeepSeek-V4F on an `mtu_mismatch` task of the main evaluation. Figure 7 is the input: the system prompt and the symptom report as the agent receives them, and the inventory excerpt that names every node, link, and loopback but not the fault. Figure 8 lists every reasoning round and every tool call with its arguments and the decisive lines of its output: three interface inspections localize the mismatched MTU on `a1:eth1`, one repair call restores it, and two re-probes confirm that the symptom is gone. Figure 9 is the verifier's reading of the network after the finish declaration, in which the probe, state, and route targets pass, the 28 re-probed baseline pairs remain reachable, and the declared node lies in the equivalence class of the injected location. The episode uses 10 tool calls over 7 model rounds and 120 s against a reference solution of 5 calls, so its tool-call ratio is 2.0.

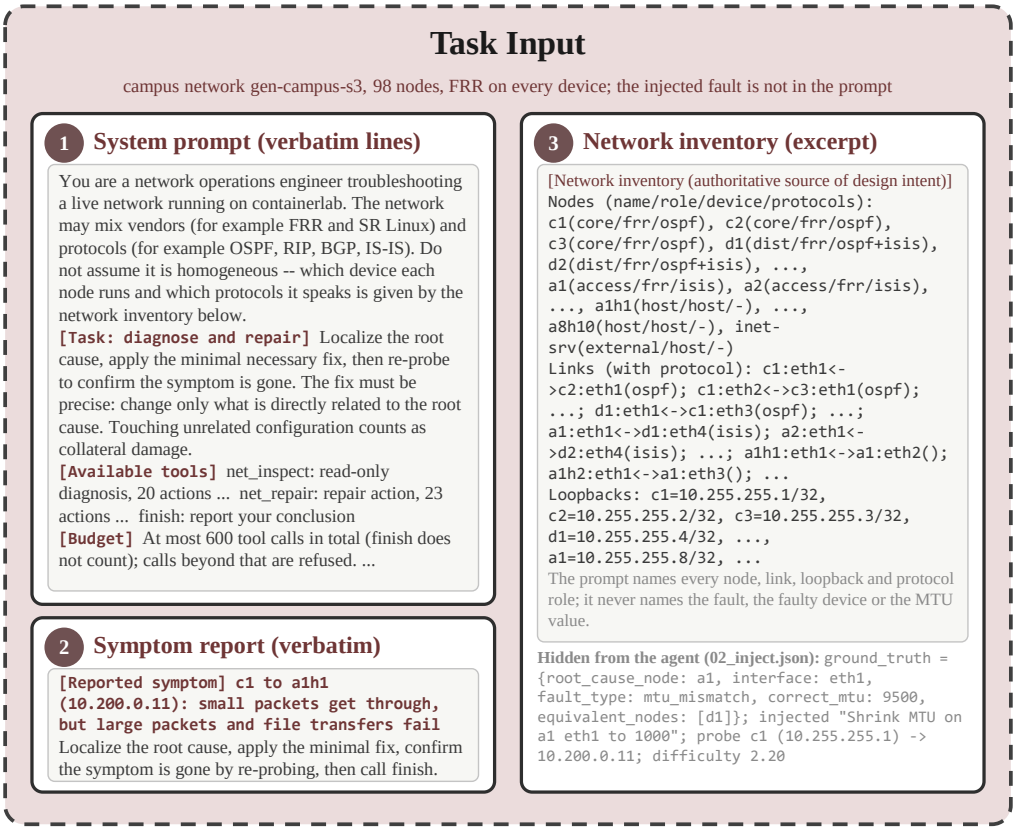


Figure 7: One episode on an `mtu_mismatch` task, part one: the task input as the agent receives it. Prompt lines are verbatim, and the injected fault stays hidden from the agent.

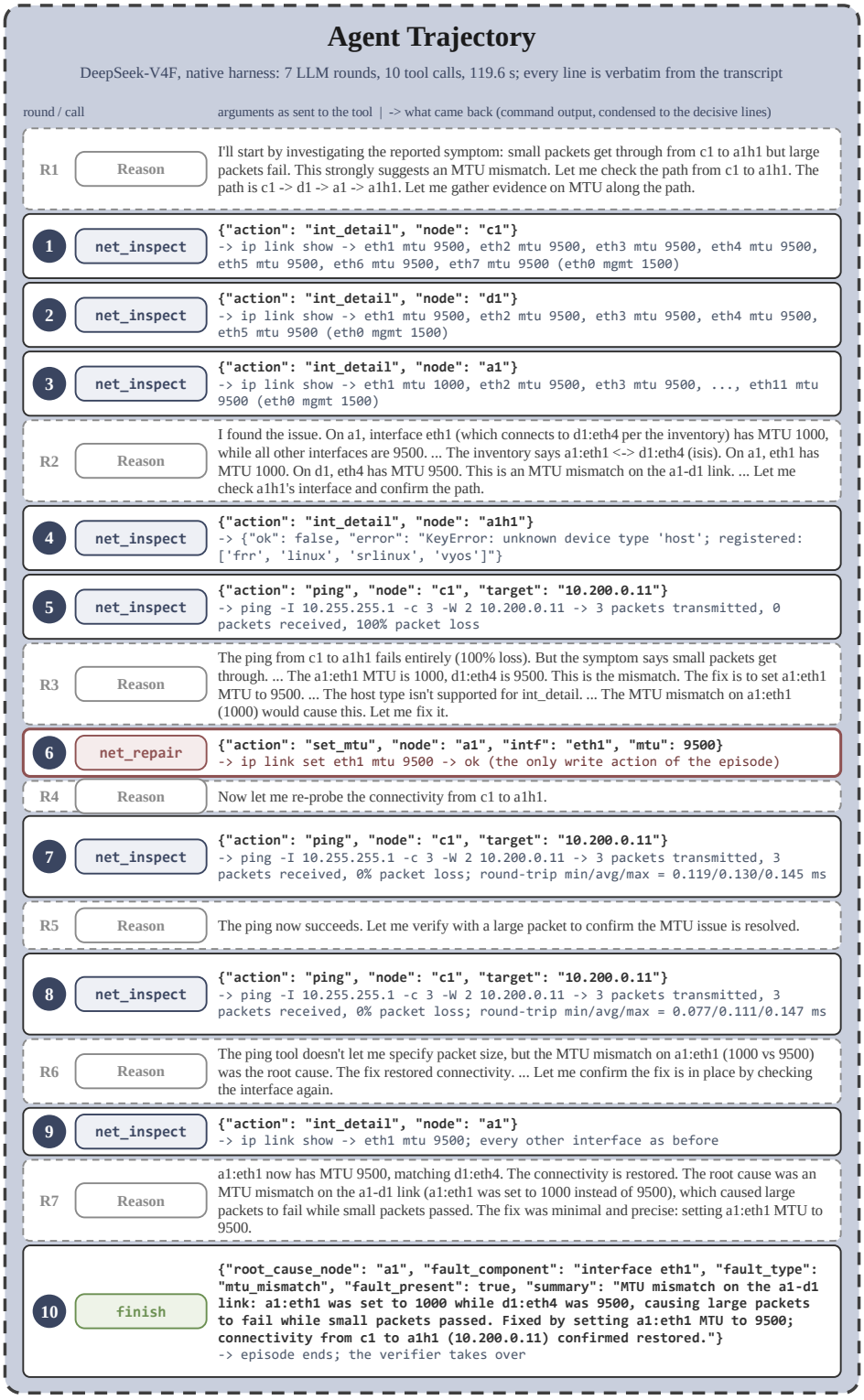


Figure 8: The same episode, part two: all seven reasoning rounds and ten tool calls of DeepSeek-V4F, with the arguments as sent and the decisive lines of each result. The trajectory is unmodified.

State Verifier

every value below is read from the running network after finish

1 Targets of the specification S_ver = (P, D, R), the safety re-probe, and the declaration

target	after injection	after repair	verdict
probe c1 -> a1h1 (P)	0/3 received, 100% loss	3/3 received, 0% loss, 0.13 ms	pass
a1:eth1 MTU (D)	1000	9500 = baseline	pass
routes c1, d1, a1 (R)	unchanged	unchanged	pass
28 baseline pairs	-	all reachable, no regression	no_collateral, quality_ok
finish: a1, mtu_mismatch	-	a1 in E(l) = {a1, d1}	hit

$\text{pass}(\tau) = 1 \quad \text{hit}(\tau) = 1 \quad \text{no_collateral}(\tau) = 1 \quad t_{\text{conv}} = 0\ \text{s} \quad |A| = 10,\ |A^*| = \lceil \log_2 5 \rceil + 2 = 5,\ \text{ref_ratio} = 2.0$

Figure 9: The same episode, part three: the verifier's reading of the live network after the finish declaration, and the resulting scores.