Evaluating Agentic Configuration Repair for Computer Networks

Rufat Asadli [1] **Benjamin Hoffman** [*1] **Ioannis Protogeros** [*1] **Laurent Vanbever** [1]

Abstract

Misconfigurations in computer networks remain a major source of critical Internet outages. Research is turning to Large Language Models (LLMs) to automate the complex, error-prone task of network configuration. However, even state-of-the-art models fail to resolve misconfigurations in large-scale, complex scenarios and often introduce new errors. In this work, we benchmark open- and closed-source LLMs augmented with formal network verification and context retrieval tools. We demonstrate that agentic architectures outperform base LLMs in *repair efficacy* (by 12% on average) and *safety* (by 17% on average), enabled by the ability to dynamically manage context and iteratively validate configuration repairs.

1. Introduction

Network misconfigurations are a leading source of Internet outages, causing a large share of downtime in enterprise and ISP networks (Janardhan, 2021; Prince, 2025). Thus, ensuring and maintaining network correctness has been a long-standing research goal, which remains largely unrealized due to the growing complexity of network infrastructure. Formal verification and synthesis have made significant strides, but remain inapplicable to many real-world, open-ended deployments due to limited and sometimes inaccurate modeling of complex network behavior (Krentsel et al., 2025). As a result, human errors continue to be the root cause of numerous impactful and costly network outages.

LLMs for network operations Following their impressive software engineering performance (Jimenez et al., 2024; Jain et al., 2024; Vero et al., 2025), LLMs stand as a more flexible alternative for automating network operations. To this end, hyperscalers have already started deploying them in production — including ByteDance's NetAssistant (Wang et al., 2024), Alibaba's BiAn (Wang et al., 2025a), and Meta's Confucius (Wang et al., 2025d). However, adoption remains limited by safety concerns: hallucinations and erroneous outputs can cause catastrophic outages.

Existing benchmarks Recent works have begun to rigorously quantify the viability of LLMs in network operations. NETARENA (Zhou et al., 2026) provides a dynamic evaluation environment for LLMs spanning basic network analysis, verification, and configuration tasks. NIKA (Wang et al., 2025c) proposes a structured methodology to use LLM agents in network troubleshooting; however, it does not support the evaluation of proposed fixes for network faults. More recently, CORNETTO (Protogeros et al., 2026) evaluates LLM-based end-to-end configuration repair across diverse protocols and topologies at scale.

Collectively, these efforts reveal that even frontier models struggle with complex network tasks under *monolithic* (i.e., single-turn) prompting. Three recurring challenges emerge: *(i)* configurations span thousands of lines where noise obscures relevant signals; *(ii)* single-shot attempts hinder error correction, causing models to stop at partial diagnoses; and *(iii)* the semantic gap between edits and their emergent *forwarding behavior* (i.e., how traffic flows through a network) makes their network-wide impact difficult to predict.

This work: Agentic repair for misconfigurations Agentic architectures show promise to address each of these limitations — for instance, agent-computer interfaces have proven important in LLM-driven software engineering (Yang et al., 2024; Wang et al., 2025b). We apply this paradigm for network configuration, with tools for dynamic context retrieval to help filter noise; iterative editing to address single-shot failures; and verifier access allowing models to safely examine network-wide reconfiguration effects without live deployment. Unlike prior agent benchmarks for networking, which focus on short diagnostics or single-command tasks (Wang et al., 2025c; Zhou et al., 2026), this setup targets end-to-end iterative configuration repair.

Building on this, we evaluate open- and closed-source LLMs on CORNETTO, equipping them with (1) dynamic context retrieval, (2) iterative search-and-replace editing, and (3) formal verification as environment feedback.

[*]Equal contribution, order determined by coin flip. [1]Department of Information Technology and Electrical Engineering, ETH Zurich. Correspondence to: Rufat Asadli <rasadli@ethz.ch>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026.

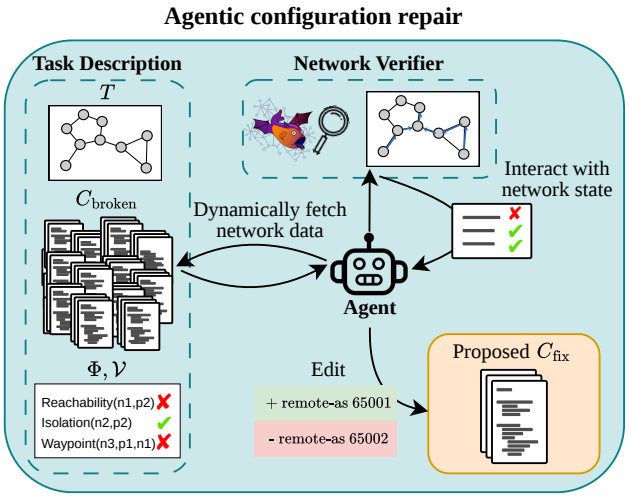


Figure 1. Overview of the agentic configuration repair pipeline. Given a task description consisting of a network topology T, broken configurations C_{broken}, and violated specifications $\mathcal{V}$, the agent dynamically retrieves relevant context of its own choice, proposes search-and-replace edits, and interacts with a verifier to see the state of unresolved specifications before submitting the final C_{fix}.

Key contributions Our summarized contributions are:

- We propose a minimal agentic environment for network configuration with tool calls for dynamic context retrieval, iterative repair, and verification feedback.
- Using our setup, we find that the agentic system noticeably improves over monolithic approaches, with 12% gain in restoring correct network behavior and 17% reduction in safety regressions *on average*.
- We show that dynamic context management and continued access to a verifier positively affect the agent's safety, while the effect on efficacy is more nuanced.

2. Problem Setting and Benchmark

Task definition We evaluate agents on CORNETTO (Protogeros et al., 2026), a benchmark for LLM-driven network configuration repair. Computer networks are governed by ***network-wide configurations***: sets of distributed, per-router text files that specify how each router processes and forwards traffic (e.g., via protocols such as BGP and OSPF). Misconfigurations in these files can violate intended network behavior (e.g., dropping or misrouting traffic), and are notoriously hard to resolve (Mahajan et al., 2002).

The benchmark defines a misconfiguration scenario as a tuple $(T, C_{\text{gold}}, C_{\text{broken}}, \Phi)$, where T is the network topology, C_{gold} is the correct "golden" configuration, $C_{\text{broken}} = f(C_{\text{gold}})$ is a faulty configuration derived by applying a fault function f, and Φ is the set of specifications satisfied by C_{gold} (denoted as $C_{\text{gold}} \models \Phi$). A ***network specification*** $\phi \in \Phi$ captures the network's intended functionality as a boolean predicate over forwarding properties (e.g., *"Router A must be able to reach prefix 10.0.0.0/8"*, or *"Router B's traffic to the same prefix must pass through router C"*). The set of violated specifications is $\mathcal{V} = \{\phi \in \Phi \mid C_{\text{broken}} \not\models \phi\}$. Given the task description $\mathcal{I} = (T, C_{\text{broken}}, \mathcal{V})$, the system under test must repair the configuration by producing a configuration $C_{\text{fix}} = \mathcal{R}(\mathcal{I})$ such that $C_{\text{fix}} \models \Phi$.

Dataset The benchmark comprises 231 misconfiguration scenarios across real-world topologies from Topology Zoo (Knight et al., 2011), spanning 27 fault types (App. B.4) with up to 8 simultaneous faults per scenario. Networks range up to 754 nodes and 200K configuration lines. Table 3 (App. B.1) summarizes key dataset statistics. A defining characteristic is the disproportion between configuration perturbation and disruption in network behavior: faults touch less than 1% of configuration lines on average, yet disrupt up to 50% of network specifications. Thus, resolving a scenario requires navigating tens of thousands of lines within distributed configuration files to locate a handful of relevant lines, making precise context management a core challenge.

Evaluation method Proposed fixes are evaluated by simulating the forwarding behavior of C_{fix} and checking it against the ground-truth specification set Φ (Protogeros et al., 2026). [1] The performance for *efficacy* and *safety* is quantified via two primary metrics ($\in [0, 1]$), respectively

$$\text{Fix Score} = \frac{|\Phi_{\text{fixed}}|}{|\Phi_{\text{fixed}}| + |\Phi_{\text{unfixed}}| + |\Phi_{\text{regressed}}|},$$

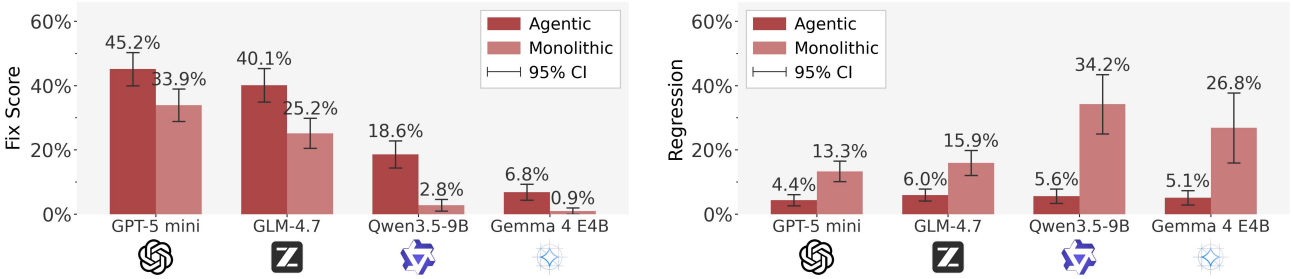
$$\text{Regression} = \frac{|\Phi_{\text{regressed}}|}{|\Phi_{\text{fixed}}| + |\Phi_{\text{unfixed}}| + |\Phi_{\text{regressed}}|},$$

where Φ_{fixed} are initially violated specifications restored by the fix, $\Phi_{\text{regressed}}$ are previously healthy specifications broken by the fix, and Φ_{unfixed} are violations that remain unresolved. A repair is considered strictly successful only if it achieves a perfect fix score with zero regressions. This evaluation mirrors test-driven development in software engineering (Jimenez et al., 2024) where the specification set Φ is a collection of unit tests that can be fixed or regressed.

3. Agentic Setup

The limitations of monolithic prompting motivate three design considerations for an agentic network configuration repair system: (1) selective retrieval of relevant configuration context based on the problem description, (2) iterative

[1] The network verifier Batfish (Fogel et al., 2015) is used along with Config2Spec (Birkner et al., 2020) to simulate network behavior and extract sets of satisfied and ground-truth specifications.



(a) Fix Score (↑ *is better*) *(b)* Regression Rate (↓ *is better*)

Figure 2. Performance of the agentic pipeline (with context retrieval, without iterative verification) against monolithic baselines (in %).

editing with rollback to correct prior mistakes, and (3) interaction with a verifier to validate intermediate repairs.

We realize these design goals by developing a ReAct (Yao et al., 2023)-style agent equipped with custom tools for selective configuration retrieval, iterative editing, and verification feedback via Batfish (Fogel et al., 2015). Descriptions of these custom-built tools are available in App. C.2. Given an input $\mathcal{I} = (T, C_{\text{broken}}, \mathcal{V})$, the agent is not provided with any task information initially. Instead, it retrieves the network topology T, the set of violated specifications $\mathcal{V}$, and the configuration files C_{broken} *on demand* (App. C.1). At each step, the agent reasons about the current state, selects a tool action, and observes the result. The agent is given a budget of N steps; if no solution is submitted by then, the existing work is submitted instead (Yang et al., 2024).

Dynamic context management Feeding the full network-wide configuration, while technically possible with current context limits, is shown to be ineffective at scale (Protogeros et al., 2026; Zhang et al., 2026). Instead, we allow the agent to list existing routers, retrieve individual configuration files, and inspect the network topology, guided by the violated specifications describing the undesired network behaviour.

Iterative repair The agent proposes configuration edits using a dedicated tool for code-patch application. The edits are performed on a working copy of C_{broken}, which the agent can inspect after each modification. This allows the agent to iteratively address faults, diagnosing and repairing each issue before moving to the next, rather than in a single pass.

Verification as feedback Formal verification provides a key source of rigour in LLM-driven software engineering workflows (Zeng et al., 2026). Similarly, at any point during the repair process, we allow the agent to invoke the CORNETTO verification pipeline as a tool. It runs in the current configuration state and returns the same three-fold specification decomposition used for the final evaluation (Sec. 2), enabling the agent to diagnose remaining conflicts, guide subsequent edits, or decide when to submit a solution.

4. Preliminary Evaluation

We first demonstrate and compare the performance of our agentic approach against monolithic LLMs in CORNETTO and subsequently analyze the effects of our different design choices on both agent performance and behavior.

Main results We evaluate four open- and closed-source LLMs (GPT-5 MINI, GLM-4.7, QWEN3.5-9B, GEMMA 4 E4B; details in App. B.2) on all 231 misconfiguration scenarios using our agentic setup. To ensure a fair comparison to the monolithic baseline, we initially *disable* verification feedback, using it only at submission for scoring. All other tools are available with a budget of 30 iterations per task.

Our agentic approach consistently outperforms monolithic LLMs in both fix score and regression rate (Fig. 2). The benefit is especially pronounced for open-source models, which perform poorly in the monolithic setting but see their fix scores increase by up to 7× under the agentic framework, while their regression rates drop from as high as 34.2% to under 6%. This suggests that agentic scaffolding disproportionately compensates for weaker base-model capability, while also stabilizing their safety behavior.

Design effects We isolate two design choices, access to iterative verification feedback and dynamic context retrieval, ablating each independently (Tab. 1). For these experiments, we focus on GPT-5 MINI and QWEN3.5-9B, given they are the best performing closed- and open-source models from our main evaluation, respectively. First, *enabling* iterative verifier feedback benefits QWEN3.5-9B, achieving higher fix scores and lower regressions than the agentic setup without feedback. For GPT-5 MINI, it cuts regressions, but also decreases the fix score, suggesting that additional verification calls make the model more conservative in its edits.

Next, while iterative verifier feedback is enabled, we test the impact of context retrieval. GPT-5 MINI performs best at 49% fix score when the entire problem description $(T, C_{\text{broken}}, \mathcal{V})$ is *prefilled* in its context window a priori rather than *dynamically* retrieved. We attribute this to the

Table 1. Effects of two key design choices in the agentic pipeline: (1) verifier feedback and (2) context retrieval tools.

Component	Setting	Fix ↑	Regr. ↓
Verifier Feedback			
GPT-5 MINI	Without feedback	**45.2**	4.4
	With feedback	40.8	**1.3**
	Δ	-4.4	-3.1
QWEN3.5-9B	Without feedback	18.6	5.6
	With feedback	**20.0**	**2.9**
	Δ	+1.4	-2.7
Context Retrieval			
GPT-5 MINI	Prefilled retrieval	**49.1**	1.3
	Dynamic retrieval	40.8	1.3
	Δ	-8.3	0.0
QWEN3.5-9B	Prefilled retrieval	14.8	8.3
	Dynamic retrieval	**20.0**	**2.9**
	Δ	+5.2	-5.4

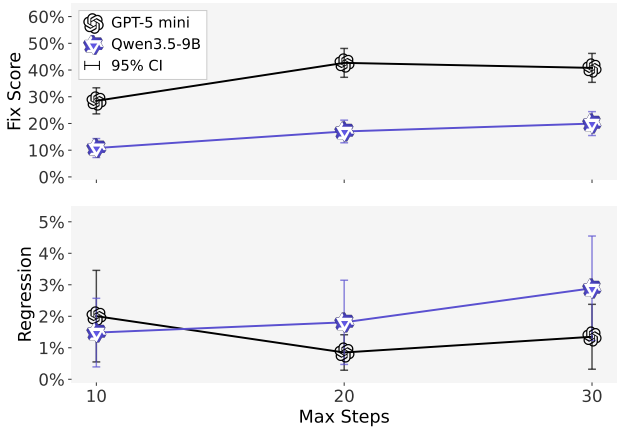


Figure 3. Effects of different maximum step budgets on agents (↑ *is better* for Fix Score, ↓ *is better* for Regression).

model's strong long-context capabilities, which allow it to effectively process the full scenario in a single pass, reducing the benefit of spending steps on context retrieval versus spending them on verifier feedback and editing. QWEN3.5-9B, despite its nominally large context window, does not benefit from prefilled inputs, as smaller models often struggle to extract relevant signals from large contexts (Zhang et al., 2026). Instead, with targeted context retrieval, it exhibits a higher fix score (20%) and lower regression (3%).

Furthermore, with all tools available, we evaluate agent performance across different maximum step budgets (Fig. 3). While more iterations generally improve fix scores, the effect on regressions is more nuanced: QWEN3.5-9B's regression rate actually rises with longer budgets, likely reflecting the model's difficulty in retaining coherence over long *interaction histories* (App. B.3). Beyond efficacy, this analysis informs the cost profile of the agentic pipeline — compute for self-hosted open-source models and tokens for metered

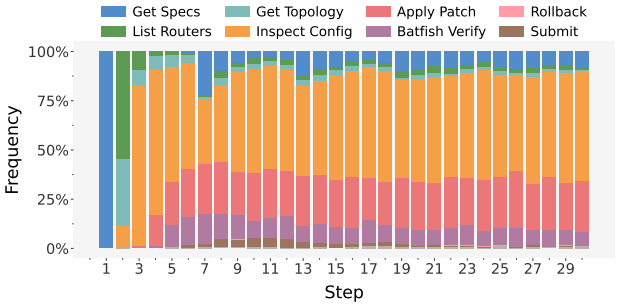


Figure 4. The frequency of the tools called at each iteration across all models over all task instances per model.

proprietary ones. Since multi-turn inference is inherently more expensive than single-turn prompting, identifying the step budget at which fix scores saturate is essential for deploying agentic repair at reasonable cost (App. A.3).

Agent trajectory To better understand an agent's decision-making process, we evaluate their tool-calling behavior across 30 steps (Fig. 4), revealing a structured workflow: agents consistently begin by gathering specifications and inspecting the network topology, before transitioning to the patch application and verification loop in the middle steps. Per-model trajectories (App. A.2) further show that models tend to exhaust their budget rather than converging early. Future work could explore tool associations and more granular feedback to better understand agent decision-making.

5. Limitations and Future Work

We show the benefits of our agentic approach over base LLMs for network configuration repair, improving efficacy and safety. However, the interactions among design choices introduce trade-offs that necessitate further investigation:

LLM-network interfaces Unlike offline verification, richer interfaces, such as live routing tables and packet traces, would expand coverage to transient and performance-related faults that static analysis alone cannot diagnose.

Safety guardrails More proactive guardrails to pre-screen edits against known invariants, along with human-in-the-loop approval for high-risk changes, could further improve the viability of agentic repair for real-world deployments.

Fine-tuning Iterative decision-making in agentic systems also impacts training methodologies: there is growing interest in RL-based recipes that apply step-level learning, assigning feedback to intermediate steps rather than final outcomes only (Zha et al., 2025; Xu et al., 2026). This can be further explored using small open-source models as efficient, privacy-preserving alternatives to proprietary ones.

Impact Statement

This paper aims to contribute towards the safe adoption of LLM-based agents for automated network management. A potential social outcome of our work is improved reliability of network operations, contributing to the resilience of Internet-reliant critical services — network misconfigurations are currently a major cause of Internet outages.

References

Bates, T., Chen, E., and Chandra, R. Bgp route reflection: An alternative to full mesh internal bgp (ibgp). RFC 4456, RFC Editor, April 2006. URL https://www.rfc-editor.org/rfc/rfc4456.txt.

Birkner, R., Drachsler-Cohen, D., Vanbever, L., and Vechev, M. Config2Spec: Mining network specifications from network configurations. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pp. 969–984, Santa Clara, CA, February 2020. USENIX Association. ISBN 978-1-939133-13-7. URL https://www.usenix.org/conference/nsdi20/presentation/birkner.

Fogel, A., Fung, S., Pedrosa, L., Walraed-Sullivan, M., Govindan, R., Mahajan, R., and Millstein, T. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 469–483, Oakland, CA, May 2015. USENIX Association. ISBN 978-1-931971-218. URL https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/fogel.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Janardhan, S. More details about the October 4 outage. Engineering at Meta Blog, 2021. URL https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/. Accessed: 2026-04-29.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Knight, S., Nguyen, H., Falkner, N., Bowden, R., and Roughan, M. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, October 2011. ISSN 0733-8716. doi: 10.1109/JSAC.2011.111002. Funding Information: The authors would like to acknowledge the support of the Australian Research Council through grants DP0985063 and DP110103505, and two Australian Postgraduate Awards.

Krentsel, A., Ye, O., Tafoya, A., Ma, X., Ratnasamy, S., and Shaikh, A. Towards accessible model-free verification. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks*, HotNets '25, pp. 210–217, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400722806. doi: 10.1145/3772356.3772380. URL https://doi.org/10.1145/3772356.3772380.

Mahajan, R., Wetherall, D., and Anderson, T. Understanding bgp misconfiguration. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pp. 3–16, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 158113570X. doi: 10.1145/633025.633027. URL https://doi.org/10.1145/633025.633027.

Prince, M. Cloudflare outage on November 18, 2025. Cloudflare Blog, 2025. URL https://blog.cloudflare.com/18-november-2025-outage/. Accessed: 2026-04-29.

Protogeros, I., Asadli, R., Hoffman, B., and Vanbever, L. Benchmarking llm-driven network configuration repair, 2026. URL https://arxiv.org/abs/2604.22513.

Shamim, F., Aziz, Z., Liu, J., and Martey, A. *Troubleshooting IP Routing Protocols*. CCIE Professional Development Series. Cisco Press, 2002. ISBN 9780133034684. URL https://www.ciscopress.com/store/troubleshooting-ip-routing-protocols-ccie-professional-9780133034684.

Vero, M., Mündler, N., Chibotaru, V., Raychev, V., Baader, M., Jovanović, N., He, J., and Vechev, M. Baxbench: Can llms generate correct and secure backends?, 2025. URL https://arxiv.org/abs/2502.11844.

Wang, C., Zhang, X., Lu, R., Lin, X., Zeng, X., Zhang, X., An, Z., Wu, G., Gao, J., Tian, C., Chen, G., Liu, G., Liao, Y., Lin, T., Cai, D., and Zhai, E. Towards llm-based failure localization in production-scale networks. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, pp. 496–511, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400715242. doi: 10.1145/3718958.3750505. URL https://doi.org/10.1145/3718958.3750505.

Wang, H., Abhashkumar, A., Lin, C., Zhang, T., Gu, X., Ma, N., Wu, C., Liu, S., Zhou, W., Dong, Y., Jiang, W., and Wang, Y. NetAssistant: Dialogue based network

diagnosis in data center networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 2011–2024, Santa Clara, CA, April 2024. USENIX Association. ISBN 978-1-939133-39-7. URL https://www.usenix.org/conference/nsdi24/presentation/wang-haopei.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. Openhands: An open platform for ai software developers as generalist agents, 2025b. URL https://arxiv.org/abs/2407.16741.

Wang, Z., Cornacchia, A., Sacco, A., Galante, F., Canini, M., and Jiang, D. A network arena for benchmarking ai agents on network troubleshooting, 2025c. URL https://arxiv.org/abs/2512.16381.

Wang, Z., Lin, S., Yan, G., Ghorbani, S., Yu, M., Zhou, J., Hu, N., Baruah, L., Peters, S., Kamath, S., Yang, J., and Zhang, Y. Intent-driven network management with multi-agent llms: The confucius framework. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, pp. 347–362, New York, NY, USA, 2025d. Association for Computing Machinery. ISBN 9798400715242. doi: 10.1145/3718958.3750537. URL https://doi.org/10.1145/3718958.3750537.

Xu, R., Chen, J., Ye, J., Wu, Y., Yan, J., Yang, C., and Yu, H. Incentivizing agentic reasoning in llm judges via tool-integrated reinforcement learning, 2026. URL https://arxiv.org/abs/2510.23038.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Zeng, L., Che, F., Huang, X., Ye, F., Xu, X., Yuan, B., and Fu, J. Veriequivbench: An equivalence score for ground-truth-free evaluation of formally verifiable code, 2026. URL https://arxiv.org/abs/2510.06296.

Zha, K., Gao, Z., Shen, M., Hong, Z.-W., Boning, D. S., and Katabi, D. Rl tango: Reinforcing generator and verifier together for language reasoning, 2025. URL https://arxiv.org/abs/2505.15034.

Zhang, Q., Hu, C., Upasani, S., Ma, B., Hong, F., Kamanuru, V., Rainton, J., Wu, C., Ji, M., Li, H., Thakker, U., Zou, J., and Olukotun, K. Agentic context engineering: Evolving contexts for self-improving language models, 2026. URL https://arxiv.org/abs/2510.04618.

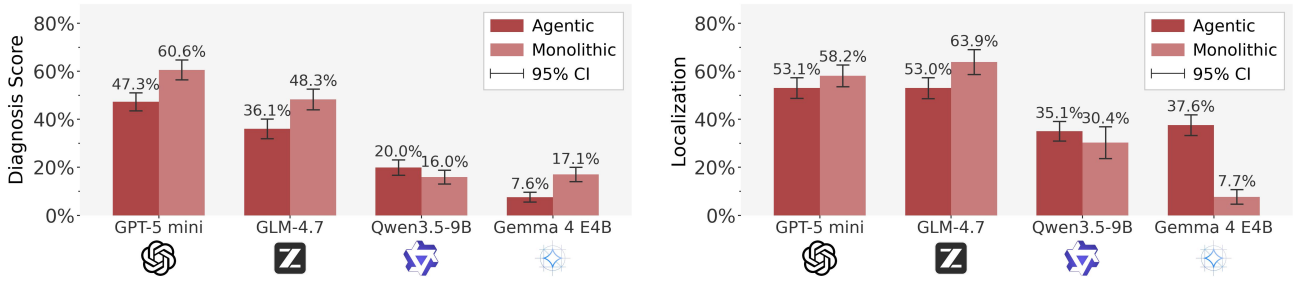
Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Zhou, Y., Ruan, J., Wang, E. S., Fouladi, S., Yan, F. Y., Hsieh, K., and Liu, Z. Netarena: Dynamic benchmarks for ai agents in network automation, 2026. URL https://arxiv.org/abs/2506.03231.

A. Additional Results

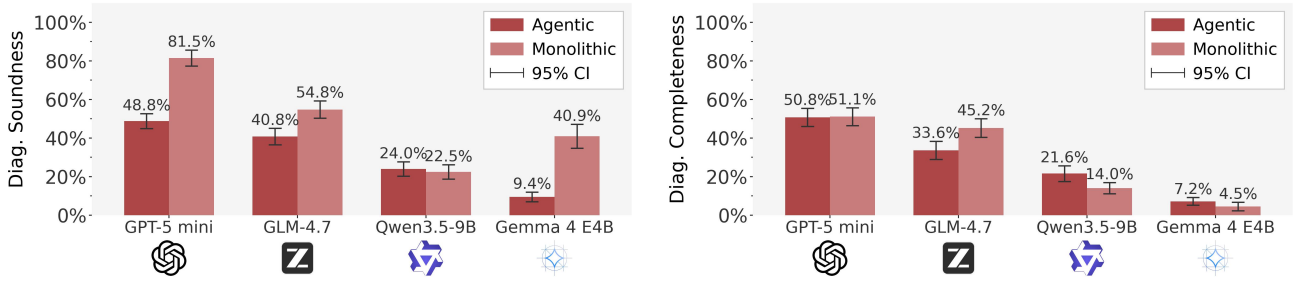
In this part, we report additional results that complement the main evaluation: the diagnosis and localization scores comparing agentic and monolithic setups (App. A.1), a per-model decomposition of agent tool-calling trajectories (App. A.2), and a cost analysis of the agentic pipeline against monolithic baselines (App. A.3).

A.1. Diagnosis and Localization



(a) Diagnosis (↑ *is better*) *(b)* Localization (↑ *is better*)

Figure 5. Comparison of agentic and monolithic LLMs in terms of diagnosis score and root-cause localization performance (in %).



(a) Diagnosis Soundness (↑ *is better*) *(b)* Diagnosis Completeness (↑ *is better*)

Figure 6. Comparison of agentic and monolithic LLMs in terms of diagnosis soundness and completeness performance (in %).

Beyond reporting the fix score and regression rate for the agents against monolithic baselines (Sec. 4), we present additional metrics in this part: the diagnosis and localization scores (Fig. 5). Following CORNETTO, beyond editing configuration files, we require the agents to provide a textual diagnosis of their solution and to list the files they deem problematic. We report diagnosis scores using an LLM-as-a-Judge method (Zheng et al., 2023), in which three LLM judges receive the ground-truth problem setting and evaluate how well the agents identify the *misconfigurations* (see App. B.3 for details). We compute the localization as a classification score, measuring how well the agents identify the *misconfigured files*.

Interestingly, while the agentic setup generally outperforms the monolithic baselines in other metrics, agents tend to achieve lower diagnosis scores. We link this with the diagnosis soundness and completeness patterns shown in Fig. 6. The soundness term measures the proportion of diagnosis claims that are correct (i.e., precision), whereas completeness stands for the proportion of true faults that were actually identified (i.e., recall). We observe that agents appear to adopt a more aggressive behaviour in identifying true positives, yielding comparable recall, but at the cost of lower precision relative to monolithic models. The higher rate of false positives, potentially arising from hallucinations, introduces noise into the agents' overall fault diagnoses, ultimately reducing their diagnostic accuracy.

A.2. Agent Trajectory

In Figure 7, we decompose the per-step agent trajectory, aggregated over all models, into separate views for each agent. Agents generally tend to exhaust all the allocated maximum step budgets in a single task, although GEMMA 4 E4B uses roughly half of its budget only, given its limited capacity to retain reasonable performance over a wider interaction horizon—following a very similar tool-calling distribution as in the aggregated case (Fig. 4). Notably, agents spend most of the iterations inspecting file contents. On average, they utilize the verifier feedback 2-3 times per task, opening a potential

discussion for verification mechanisms with more powerful feedback signals. In future work, we believe that further analysis of agent behavior — more specifically, patterns in the order of tool usage across interaction trajectories — can yield valuable insights into agentic reasoning.

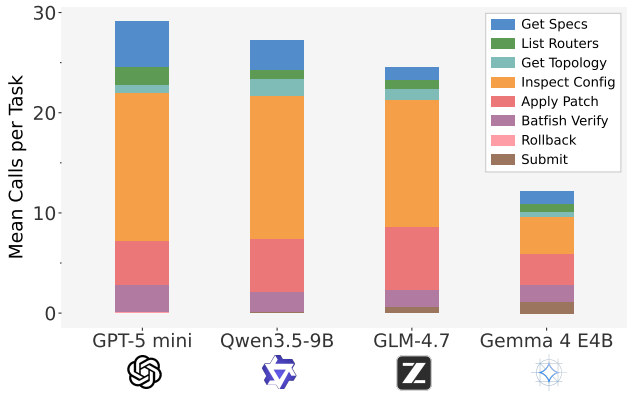
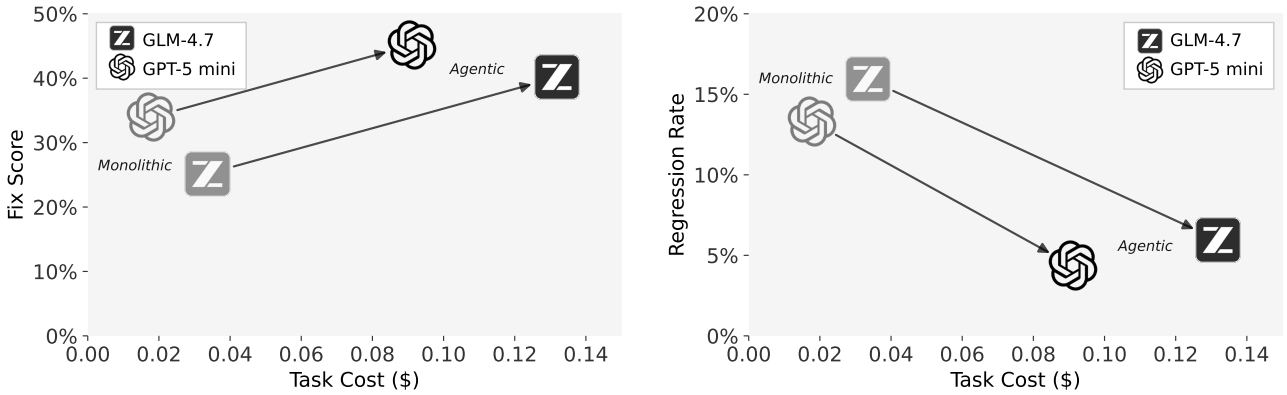


Figure 7. The frequency of the tools called by each model in a task on average.

A.3. Cost-Performance Trade-off

The superior performance of the agentic pipeline comes at a higher inference cost. As shown in Fig. 8, with a budget of 30 maximum steps, GPT-5 MINI spends $0.091 per task against $0.018 for the monolithic baseline (roughly a 5× increase). Similarly, the average per-task cost of running GLM-4.7 in the agentic setup increases by around 4× from $0.034 to $0.132. Therefore, as previously emphasized in Sec. 4, this motivates identifying the optimal step budget that recovers most of the benefit as key to controlling the cost of agentic repair in practice.



(a) Fix Score (↑) & Cost (↓)

(b) Regression Rate (↓) & Cost (↓)

Figure 8. Comparison of agentic and monolithic LLMs in terms of fix score and regression rate, against the per task cost (in $). Model identifiers, distinguished by the original and faded logos, stand for the agentic and monolithic setups, respectively.

Example cost-step analysis In Table 2, we compare the monolithic baseline to the agentic setup at different iteration budgets and pipeline modes. With iterative verifier feedback enabled, at just 10 steps, GPT-5 MINI already reaches a 28.5% fix score (vs. 33.9% monolithic), while cutting the regression rate from 13.3% to 2.0% — at a cost of only $0.023, barely above the monolithic baseline. By 20 steps, the agentic pipeline surpasses the baseline by 10% with a fix score of 42.7% and holds regression below 1%, for roughly $0.050 per task. With feedback mode, going from 20 to 30 steps yields a very similar cost with a marginal drop in performance metrics, which might be due to inherent randomness in model responses and also changes in the tool-calling trajectory. This points to diminishing returns beyond 20 steps, as the most noticeable gains occur in the first 10-20 iterations. However, this observation is based on a three-tier step budget analysis and a single model. The precise inflection point may shift with more granular step increments, different models, or task-specific tools.

Table 2. Cost–performance trade-off for GPT-5 MINI across pipeline configurations. Against the monolithic baseline, we demonstrate the performance of the agentic setup with and without access to the verifier feedback as a tool.

Pipeline	Steps	Fix Score (%)	Regression (%)	Cost ($)
Monolithic	1	33.9	13.3	0.018
Agentic w/o feedback	30	45.2	4.4	0.091
Agentic w/ feedback	10	28.5	2.0	0.023
Agentic w/ feedback	20	42.7	0.9	0.050
Agentic w/ feedback	30	40.8	1.3	0.049

Lastly, with verifier feedback disabled, increasing the step budget to 30 iterations almost doubles the cost, with the fix score improving to 45.2% but the regression rate decreases by 3.1 percentage points. Comparing the two pipeline variants at 30 steps reveals the role of the verifier: when verifier feedback is disabled, the agent can only utilize the context retrieval tools, which return more information-rich context about the network state (e.g., configuration files, topology, specification). These components consume additional tokens at each iteration and thus compound the total cost.

B. Experimental Details

To support the results in Sec. 4, we detail the design choices underlying our experiments: the statistics of the CORNETTO benchmark test set (App. B.1), the model selection spanning a capability spectrum (App. B.2), and the pipeline configurations covering LLM-as-a-Judge diagnosis, interaction-history management, and context sampling (App. B.3). Finally, we include a comprehensive list of benchmark misconfigurations in App. B.4 for completeness.

B.1. Dataset Statistics

To directly compare our setup to the monolithic baseline, we evaluate the agents on the CORNETTO test set, which is modeled after frequently observed real-world outages. As mentioned earlier, it comprises 231 misconfiguration scenarios across topologies from the Topology Zoo, spanning 27 fault types with up to 8 simultaneous faults per scenario. Fault injection mechanisms are minimal yet highly disruptive in impact, requiring agents to have a solid domain knowledge to reason about accurately locating and repairing the faults throughout the iterative process.

Table 3. CORNETTO dataset statistics (Protogeros et al., 2026).

	Metric	Mean	Max
Topology	Nodes (#)	86.4	754
	Config lines	16.1K	200.0K
	Data plane predicates	12.7K	598.0K
Fault impact	Routers affected	5.5	20
Impact (%)	LoC% changed	0.44	5.93
	Routes% changed	6.34	57.3
	Predicates% changed	6.82	49.8

B.2. Model Selection

To show the feasibility of our minimal agentic setup over monolithic architectures, we first conduct a focused comparison using four models spanning a range of capabilities: GPT-5 MINI and GLM-4.7 as strong proprietary and closed-source representatives, alongside QWEN3.5-9B and GEMMA 4 E4B as smaller open-source alternatives. Rather than aiming for exhaustive model coverage, this selection allows us to test whether the agentic benefit holds across a capability spectrum — from models that already perform reasonably on CORNETTO to those that struggle in the monolithic setting. As we show in Sec. 4, the pattern is consistent: agentic scaffolding improves fix scores and reduces regressions for all selected models, with disproportionately large gains for open-source models of lower capacities. Lastly, the agents are configured with the following hyperparameters across all experiments: (1) `max_tokens` of $40,000$, (2) `top-`p of 0.95, and (3) `temperature` of 0.7.

B.3. Pipeline Configurations

Across our agentic experiments, we adhere to the following pipeline configurations.

LLM-as-a-Judge diagnosis Beyond fix score and regression rate, which capture direct repair performance, we also evaluate models' diagnostic and root-cause identification abilities. For this, we adopt an LLM-as-a-Judge (Zheng et al., 2023) approach in which three LLMs (GPT-5.1, CLAUDE 4.5 OPUS, and GEMINI 2.5 PRO) independently assess the soundness and completeness of each model's solution trajectory. We follow a multi-judge evaluation mechanism to avoid potential biases. More importantly, each judge receives the ground truth misconfiguration (i.e., the list of injected faults), meaning that agent diagnoses are rewarded only when consistent with ground truth.

Dynamic interaction history As in prior work (Yang et al., 2024), in our experiments, we grant agents access to the full running chat history at each step. In general, each iteration produces a thought-action-observation triple that is appended to the conversation. As the chat history grows, older tool outputs, particularly verbose configuration dumps and specification lists, can consume a significant portion of the context window. Following the windowing strategy from SWE-agent, we truncate observations older than the most recent 5 steps, replacing them with a short placeholder while preserving the agent's own reasoning (i.e., thought and action messages) in full. This allows the model to recall its prior decisions without accumulating additional token cost of retaining large tool outputs that it has already processed. However, for low capacity models, it might be harder to retain the accumulating interaction history, as we see in the case of QWEN3.5-9B with marginally increasing safety regressions across longer interaction cycles (Fig. 3).

Context sampling Baseline experiments from CORNETTO follow two context management strategies to handle information overflow across models with varying context windows: *oracle* mode, which provides only the configuration files that were actually modified by the fault, and *random* mode, which fills the available context window with a random subset of configurations (with a fixed randomness seed for all experiments), constrained by each model's own context window limits. Therefore, it is technically possible that a model, with a large enough context window, will have access to all configuration files available. In our agentic setup, we adopt the random sampling strategy, in which the agent can access all selected configuration files through its retrieval tools. However, unlike the monolithic baseline, which must fit all context into a single prompt, the agent retrieves configurations on demand, inspecting only those it deems relevant to the diagnosed fault. This effectively decouples context availability from context window pressure, as configurations are loaded incrementally rather than all at once.

B.4. Fault Types

The benchmark contains various misconfigurations that reflect common human errors encountered in the wild. Faults are grounded in past incidents (Janardhan, 2021), Request for Comments (RFCs) (Bates et al., 2006), and troubleshooting manuals (Shamim et al., 2002).

Table 4. Comprehensive fault catalog listing the protocols affected, the nature of the misconfiguration (Summary), and the resulting impact on the network (Expected Effect). (Protogeros et al., 2026)

Protocol / Type	Summary	Expected Effect
BGP	eBGP neighbor configured with incorrect remote AS	eBGP session reset due to ASN mismatch, cutting off inter-AS route exchange
	Administratively shut down a BGP neighbor	BGP Peering is disabled, withdrawing all prefixes learnt via the neighbor
	Node configured with incorrect local ASN	Misaligned local ASN breaks iBGP/eBGP sessions and splits the AS control plane
	Force invalid next-hop on eBGP advertisements	Outbound policy rewrites next-hop to an unreachable address, causing downstream traffic blackholes
	Remove next-hop-self from RR → client iBGP session	iBGP routes advertised to clients retain original eBGP next-hop, which may be unreachable from clients causing traffic blackholes
	Withdraw a BGP network statement from the process	Prefix is no longer originated, withdrawing reachability from downstream peers
	Remove outbound route-map from eBGP neighbor	Export policy no longer enforced, allowing infrastructure routes (loopbacks, P2P) and unintended prefixes to leak to external peers
	Swap inbound and outbound route-maps on a neighbor	Inbound filters begin applying outbound and vice versa, breaking intended import/export policy
	Leak router loopback by stripping export/import policies	ASBR originates its loopback /32 into eBGP and the peer accepts it because inbound filtering was removed
	Break RR sessions to orphan clients	iBGP sessions removed between RR and up to 5 (exclusive) clients, orphaning them from iBGP reachability
	Duplicate cluster-id across route reflectors and isolate clients on one RR	Conflicting cluster-ids cause route reflectors to drop one another's updates, stranding clients that now depend on the misconfigured RR (cf. RFC 4456, Sec. 8)
OSPF	OSPF interface cost set to extreme value	Artificially high OSPF cost diverts traffic away from the link based on alternate SPF paths
	Disable OSPF adjacency on a link	Removing the link from OSPF prevents adjacency formation and withdraws LSAs learned across it
	Node missing OSPF area membership	Router withdraws from all OSPF areas, tearing down adjacencies and LSAs
	Assign duplicate OSPF router-ID to multiple routers	OSPF adjacencies fail or LSAs rejected due to router-ID collision, fragmenting OSPF domain and blackholing traffic
IS-IS	Disable IS-IS on an intra-AS link	Removing the link from IS-IS prevents adjacency formation and withdraws LSPs learned across it
	Demote a Level-1-2 IS-IS router to Level-1	Reduces inter-area reachability by removing a backbone-capable router, risking L2 partitioning
	Assign router to wrong IS-IS area	Router in wrong area cannot form L1 adjacencies with its physical neighbors; causes partition of L1 domain and reachability loss
Addressing	Duplicate loopback IPv4 addresses	Two routers share the same loopback, risking routing loops and control-plane instability
	Link interfaces disagree on prefix length	One side of a point-to-point link uses a mismatched subnet mask, preventing adjacency formation
	Link interfaces reside in different subnets	Interfaces on a point-to-point link move to disjoint IPv4 subnets, breaking adjacency formation
Device	Remove supporting static route for advertised prefix	Advertised network disappears once the backing static route is withdrawn, causing a control-plane withdraw
Policy	Remove permit entry from prefix-list	Prefix-list no longer matches intended prefixes, causing route filtering to block previously allowed routes
	Convert BGP route-map permit clause into deny	Previously exported prefixes are now filtered, withdrawing routes from neighbors
	Lower BGP local-preference on inbound policy	Reduced local-preference makes an alternate egress the best path for affected prefixes
Redistribution	Drop BGP → OSPF redistribution on an ASBR	Internal OSPF loses external reachability because Type-5 LSAs are never originated
Security	Insert implicit deny at top of interface ACL	Ingress traffic on the protected interface is dropped before policy permits, breaking connectivity
	Insert implicit deny at top of outbound interface ACL	Egress traffic on the protected interface is dropped before policy permits, breaking connectivity

C. Prompts

C.1. Main Agentic Prompts

Agent System Prompt

A network fault has been detected. Please diagnose and repair the misconfiguration.

Start by examining the violated specifications to understand what is broken, then inspect the relevant router configurations, apply fixes, and verify.

You have a budget of {max_steps} tool calls for this task.

Agent Input Prompt

You are an expert network engineer agent. Your task is to diagnose and repair network misconfigurations by iteratively inspecting configurations, applying targeted fixes, and verifying correctness using formal data-plane analysis.

Available Tools

You interact with the network environment through tool calls. Each turn, you should output your reasoning (Thought), then a single tool call (Action).

{tool_descriptions}

Tool Call Format

These tools are NOT native function calls. You must invoke them using plain text in the EXACT format shown below. Do not use any other tool-calling syntax, function-calling format, or internal protocol. Simply output the following three lines as plain text:

```
Thought: <your reasoning about what to do next>
Action: <tool_name>
Action Input: <JSON object with parameters, or {} for no-parameter tools>
```

Examples:

```
Thought: I need to see which routers are in the network.
Action: list_routers
Action Input: {}

Thought: Let me inspect the BGP configuration of router 0_as65003_0.cfg.
Action: inspect_config
Action Input: {"router_name": "0_as65003_0.cfg"}

Thought: I found the misconfigured subnet mask. Let me fix it.
Action: apply_patch
Action Input: {"router_name": "0_as65003_0.cfg", "search": " ip address 10.0.0.1
255.255.255.0", "replace": " ip address 10.0.0.1 255.255.255.252"}

Thought: I've applied my patches. Let me verify the network state.
Action: verify
Action Input: {}

Thought: All specs are fixed with no regressions. I'll submit.
Action: submit
Action Input: {}
```

Strategy

1. Start by examining the violated specifications to understand what is broken.
2. Inspect relevant router configurations to diagnose root causes.
3. Apply targeted patches to fix the identified issues.
4. Run verification to check if specs are restored and no regressions occurred.
5. If issues remain, analyze the verification feedback and iterate.
6. If a patch introduces regressions, rollback and try a different approach.
7. Submit when satisfied or when you've exhausted reasonable repair attempts.

Important Rules

- Be precise with search blocks: they must match the config EXACTLY (whitespace, indentation, etc.).
- Prefer small, targeted patches over large rewrites.
- Always verify after applying patches before submitting.
- If verification shows regressions, consider rolling back.
- You have a limited budget of {max_steps} tool calls. Use them wisely.

C.2. Tool Stack

Agent Tool Definitions

list_routers --- List all router configuration filenames available in the network. *Returns:* A list of router filenames.

inspect_config --- Retrieve the full configuration text of a specific router. Use this to examine a router before deciding what to fix. *Returns:* The full configuration text of the specified router.

get_violated_specs --- Get the list of currently violated network specifications. These describe the gap between intended and actual behavior. *Returns:* Violated specifications with type, source, destination, and status.

get_topology --- Get the network topology (nodes and links). *Returns:* The network topology as a JSON structure.

apply_patch --- Apply a search-and-replace edit to a router configuration. The search block must appear exactly once in the config. *Returns:* Success or failure message with details.

verify --- Run formal data-plane verification on the current network state. Returns which specs are fixed, which remain broken, and any regressions. *Returns:* Verification report with fix_score, regression_rate, and per-predicate classification.

rollback --- Undo ALL patches applied since the last successful verification. Restores configs to the last verified-safe state. *Returns:* Confirmation that configs have been rolled back.

submit --- Submit the current configuration as the final solution. Call when satisfied with verification results. *Returns:* Confirmation that the solution has been submitted.

C.3. Prefilled Context Prompt

Prefilled Input Prompt

A network fault has occurred, possibly due to human error, causing discrepancies in at least one of the configuration files. The specifications that govern the network behavior have changed as a result.

Task Requirements:
1. Identify which routers must be modified to restore the correct forwarding behavior.
2. Generate the necessary modifications for each router configuration.
3. Provide your solution as explicit search-and-replace instructions for each router.

Expected Output Format:
Output must be valid YAML only, with the following top-level keys: routers, then metadata, then replacements.

Specification Semantics:
Specifications are formatted as CSV lines with columns:
Type, Source_Node, Destination_Prefix, waypoint_node, num_routes, Status

Status values:
--- broken_removed: behavior IS required but CURRENTLY MISSING. Restore it.
--- broken_added: behavior IS NOT required but CURRENTLY PRESENT. Remove it.

Invariant types: reachability, waypointing, isolation, load_balancing.

Network specifications: {preds_text}

Current network topology: {topology_text}

Router configurations:
--- START OF {filename} ---
{config}
--- END OF {filename} ---
(repeated for each router)

All network context has been provided above. The retrieval tools (list_routers, inspect_config, get_violated_specs, get_topology) are disabled --- use the information above instead.

Proceed directly to diagnosing the issue and applying patches using the Thought/Action/Action Input format.

You have a budget of {max_steps} tool calls for this task.

Begin now.