Network Management Research Group
Internet-Draft
Intended status: Informational
Expires: 1 January 2026

Y. Cui
C. Liu
X. Xie
Tsinghua University
C. Du
Zhongguancun Laboratory

30 June 2025

A Framework to Evaluate LLM Agents for Network Configuration
draft-cui-nmrg-llm-benchmark-00

Abstract

This document specifies an evaluation framework and related definitions for intent-driven network configuration using Large Language Model(LLM)-based agents. The framework combines an emulator-based interactive environment, a suite of representative tasks, and multi-dimensional metrics to assess reasoning quality, command accuracy, and functional correctness. The framework aims to enable reproducible, comprehensive, and fair comparisons among LLM-driven network configuration approaches.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://example.com/LATEST. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-cui-nmrg-llm-benchmark/.

Discussion of this document takes place on the WG Working Group mailing list (mailto:WG@example.com), which is archived at https://example.com/WG.

Source for this draft and an issue tracker can be found at https://github.com/USER/REPO.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

Network configuration is fundamental to ensuring network stability, scalability, and conformance with intended design behavior. Effective configuration requires not only a comprehensive understanding of network technologies but also advanced capabilities for interpreting complex topologies, analyzing dependencies, and specifying parameters accurately. Traditional automation approaches

such as Ansible playbooks[A2023], NETCONF[RFC6241]/YANG models[RFC7950], or program-synthesis methods-either demand extensive manual scripting or are limited to narrow problem domains[Kreutz2014]. In parallel, Large Language Models (LLMs) have demonstrated the ability to interpret natural-language instructions and generate device-specific commands, showing promise for intent-driven automation in networking. However, existing work remains fragmented and lacks a standardized way to measure whether an LLM can truly operate as an autonomous agent in realistic, multi-step configuration scenarios.

Despite encouraging results in individual subtasks, most evaluations[Wang2024NetConfEval] rely on static datasets and ad hoc metrics that do not reflect real-world complexity. As a result: - There is no common benchmark suite covering diverse configuration domains (routing, QoS, security) with clearly defined intents, topologies, and ground truth. - Existing tests seldom involve interactive environments that emulate vendor-specific device behavior or provide runtime feedback on command execution. - Evaluation metrics are often limited to simple syntactic checks or isolated command validation, failing to capture whether the intended network behavior is actually achieved.

Consequently, it is difficult to compare different LLM approaches or to identify gaps in reasoning, context-sensitivity, and error-correction capabilities[Long2025][Liu2024][Fuad2024][Lira2024]. To address these shortcomings, this document introduce *NetConfBench*, a holistic framework that provides: 1. An emulatorbased environment (built on GNS3) to simulate realistic device interactions. 2. A benchmark suite of forty tasks spanning multiple domains, each defined by intent, topology, initial state, and expert-validated ground truth. 3. Multidimensional metrics-_reasoning score_, _command score_, and _testcase score_-that evaluate an agent’s internal reasoning coherence, semantic correctness of generated commands, and functional outcomes in the emulated network.

NetConfBench aims to enable reproducible, comprehensive comparisons among singleturn LLMs, ReActstyle multiturn agents, and knowledge-augmented variants, guiding future research toward truly autonomous, intent-driven network configuration.

2. Terminology

For clarity within this document, the following terms and abbreviations are defined:

* Agent: A software component powered by an LLM that consumes a task intent, interacts with a network environment, and issues configuration commands autonomously.

* Configuration Command: A device-specific instruction (e.g., a Cisco IOS CLI line or a Juniper Junos set statement) sent by the agent to a network device.

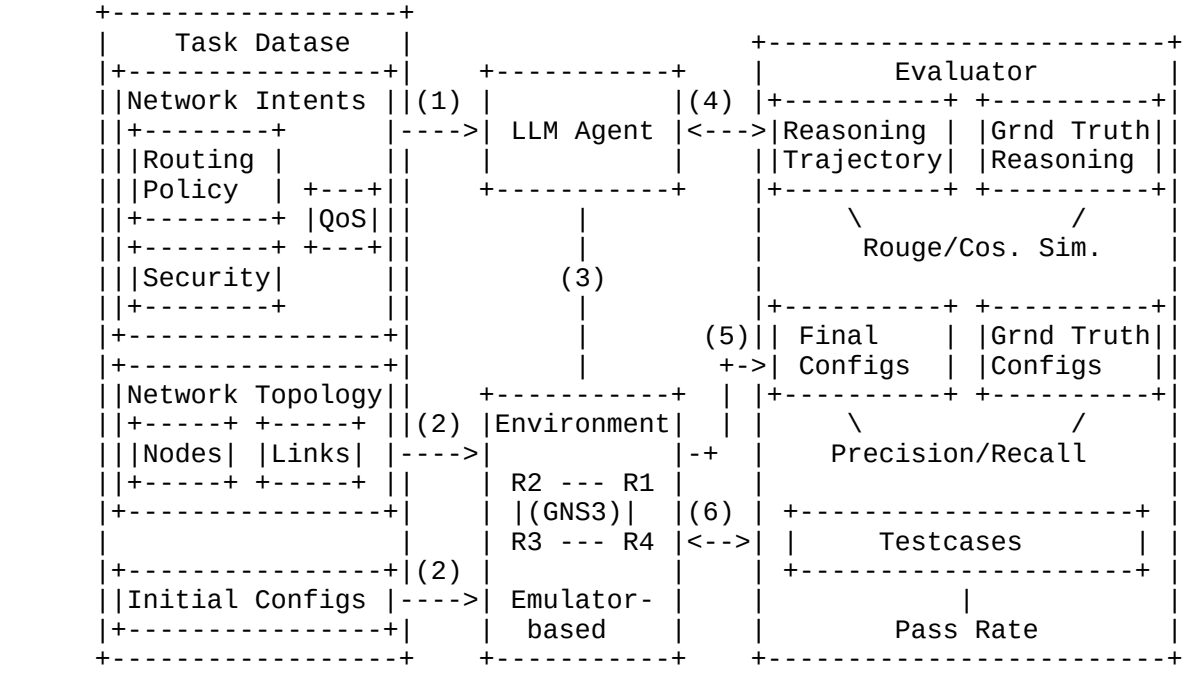
* Environment: An emulated or real network instance that exposes device status, topology information, and feedback on applied commands.

* Intent: A high-level specification of desired network behavior or objective, expressed in natural language or a structured format defined in this document.

* Task: A single evaluation unit defined by (1) a scenario category, (2) an environment topology, (3) initial device configurations, and (4) an intent. The agent is evaluated on its ability to fulfill the intent in the given environment.

* Testcase: A concrete, executable set of verification steps (e.g., ping tests, traffic-flow validation, policy checks) used to assert whether the agent’s final configuration satisfies the intent.

3. Framework Overview



Legend:
(1)Task Assignment (2)Environment Setup
(3)Interactive Task Execution (4)Reasoning Trajectory Export
(5)Final Configuration Export (6)Testcase Execution

Figure 1: The NetConfBench Framework

The proposed framework is shown in Figure 1. The flow begins with a *Task Dataset* defining network intents and topologies. The *LLM Agent* perceives the environment, reasons about required actions, and applies configuration commands. The *Environment* simulates or controls real devices, providing feedback for each action. Finally, the *Evaluator* compares the agent's outputs against ground-truth configurations and reasoning, computing scores for accuracy and completion.

3.1. Components

NetConfBench consists of four key components:

1. *Task Dataset*
   A repository of forty configuration tasks, each defined as a JSON object with:

* *Intent*: One or more natural language instructions.

* *Topology*: A list of node names and link definitions.

* *Initial Configuration*: The initial configuration state of all nodes.

* *Ground Truth Configuration*: Expert-validated CLI commands that achieve the intent.

* *Ground Truth Reasoning*: A narrative describing step-by-step logic used to derive the commands.

* *Testcases*: A set of verification procedures (e.g., _show_, _ping_, _ACL_ checks) that confirm functional intent satisfaction.

2. *Emulator Environment*
Built on GNS3, this component launches official vendor images for routers and switches, replicating realistic CLI behavior. Key interfaces include:

* *Agent-Network Interface (ANI)*: Based on the key stages commonly involved in intent-driven network configuration, we design an Agent-Network Interface to facilitate structured interactions between the LLM agent and the emulated network environment. This interface supports four core actions: get-topology, get-running-cfg, update-cfg, and execute-cmd.

  - get-topology: provides this information in a format interpretable by the LLM.

  - get-running-cfg: enables the agent to obtain the active configurations of specified devices, providing essential context for planning. subsequent updates.

  - update-cfg: allows the agent to apply new configuration commands and provides detailed feedback on their execution, including whether each command was accepted or resulted in any errors.

  - execute-cmd: accepts a device name and a command string as parameters and returns the resulting output.

* *Task Evaluation Interface*: To enable reliable and objective assessment of the LLM agent's configuration behavior, the environment provides a Task Evaluation Interface that allows the evaluation module to access relevant execution results. Specifically, this interface supports:

  - *Exporting the final configurations of all devices*: This allows for direct comparison with ground truth configurations to evaluate the correctness and completeness of the agent's output.

  - *Executing a set of predefined testcases*: These testcases are designed to verify whether the resulting network behavior accurately reflects the intended configuration objectives, as defined by the network intent.

3. *LLM Agent*
   A modular component that can be implemented with any LLM (open-source or closed-source). It interacts with the emulator via the *Agent-Network Interface* (ANI), issuing queries such as get-topology, get-running-cfg, update-cfg, and execute-cmd. Agents may use:

   * *Single-Turn Generation*: The entire reasoning and command generation in one pass.

   * *ReAct-Style Multi-Turn Interaction*: Interleaved reasoning and actions, with runtime feedback guiding subsequent steps.

   * *External Knowledge Retrieval*: (Optional) Queries to a command manual to resolve vendor-specific syntax.

4. *Evaluator*
   Computes three core metrics for each task:

   * *Reasoning Score (S_reasoning)*:

     - Embedding-based cosine similarity between the agent's reasoning trace and the ground truth reasoning.

     - Ranges from 0 to 1.

   * *Command Score (S_command)*:

     - Hierarchical diff of final vs. initial router configurations (using Python's ciscoconfparse).

- Wildcard matching ignores non-essential identifiers (e.g., ACL numbers).

- Compute precision = (correctly generated commands / total generated) and recall = (correctly generated / ground truth commands).

- S_command is the harmonic mean of precision and recall, ranging from 0 to 1.

* *Testcase Score (S_testcase)*:

  - Portion of testcases passed in the emulated environment.

  - Fine-grained sub-intents (per device) each correspond to a testcase.

  - S_testcase is the testcase pass rate, defined as the proportion of passed testcases among all defined testcases.

3.2. Workflow

The evaluation workflow for each task proceeds through six stages:

1. *Task Assignment*
   NetConfBench selects a task from the JSON dataset and provides only the high-level intent(s) to the LLM agent.

2. *Environment Setup*
   The framework instantiates a GNS3 topology based on the task's topology and applies the startup-config to each device. Once the emulated network reaches a stable state, control transfers to the agent.

3. *Interactive Execution*
   The LLM agent receives the partial prompt containing:

   * The API specification for get-topology, get-running-cfg, update-cfg, and execute-cmd.

   * The natural language intent.

   * (Optionally) Device model/version hints.
     The agent issues a sequence of API calls; for single-turn agents, it outputs reasoning followed by a batch of CLI commands. For multi-turn agents, it alternates reasoning traces and API calls.

4. *Reasoning Trajectory Export*
   After execution completes (agent signals "task done" or after a predefined command budget), NetConfBench captures the entire reasoning log:

   * For single-turn: the reasoning paragraph embedded in the LLM’s output.

   * For ReAct: an auxiliary summarization LLM condenses the interleaved reasoning and actions into a single coherent trace.

5. *Final Configuration Export*
   The framework uses the Task Evaluation Interface to extract the final running configs from each device.

6. *Testcase Execution and Scoring*

   * *Command Score:* Hierarchical diff against ground truth commands.

   * *Testcase Score:* Execute each testcase in sequence; record pass/fail.

   * *Reasoning Score:* Compute embedding similarity between the agent’s reasoning trace and ground truth reasoning.

The final per-task score is typically reported as a tuple (S_reasoning, S_command, S_testcase). Aggregate results across the forty tasks enable comparisons among LLMs and interaction strategies.

4. Data Model

This section specifies the JSON schemas and interface conventions used to represent tasks and to enable structured interaction between the LLM agent and the emulated environment.

4.1. Task Definition Schema

Each configuration task is defined as a JSON object with the following structure:

```
{
  "task_name": "Static Routing",
  "intents": [
    "NewYork: create a static route pointing to the Loopback0 on
    Washington, traffic should pass the 192.168.1.0 network.",
    "NewYork: create a backup static route pointing to the Loopback0
    on Washington, administrative distance should be 100."
    ...
  ],
  "topology": {
    "nodes": ["NewYork", "Washington"],
    "links": [
      "NewYork S0/0 <-> Washington S0/0 ",
      "NewYork S0/1 <-> Washington S0/1"
    ]
  },
  "startup_configs": {
    "NewYork": "!\r\nversion 12.4\r\nservice timestamps
    debug datetime msec\r\n...",
    "Washington": "!\r\nversion 12.4\r\nservice timestamps
    debug datetime msec\r\n...",
  },
  "ground_truth_configs": {
    "NewYork": [
      "ip route 2.2.2.0 255.255.255.252 192.168.1.2",
      "ip route 2.2.2.0 255.255.255.252 192.168.2.2 100"
    ],
    ...
  },
  "ground_truth_reasoning": "NewYork to Washington Loopback
  (primary path): add a static route for Washington's
  Loopback0 network (2.2.2.0/30) pointing to the
  next-hop 192.168.1.2...",
  "testcases": [
    {
      "name": "Static Route from NewYork to Washington",
      "expected_result": {
        "protocol": "static",
        "next_hop": "192.168.1.2"
      }
    },
    ...
  ]
}
```

4.2. Agent-Network Interface (ANI)

The Agent-Network Interface defines the minimal API primitives necessary for intent-driven configuration. Each primitive uses JSON-RPC style request/response with the following methods:

1. *get-topology*

   * *Request*:

     ```
     {
       "method": "get-topology",
       "params": {
         "devices": ["R1", "R2", ...]
       }
     }
     ```

   * *Response*:

     ```
     {
       "topology": {
         "nodes": [...],
         "links": [...]
       }
     }
     ```

   * *Description*: Returns the full topology for the specified subset of devices. If "devices" is empty or omitted, returns the entire topology.

2. *get-running-cfg*

   * *Request*:

     ```
     {
       "method": "get-running-cfg",
       "params": {
         "device": "R1"
       }
     }
     ```

   * *Response*:

```
{
  "running_config": "
   interface Gig0/0
   ip address 192.168.1.1 255.255.255.255
   ...
  "
}
```

* *Description*: Retrieves the active (running) configuration of the specified device.

3. *update-cfg*

* *Request*:

```
{
  "method": "update-cfg",
  "params": {
    "device": "R1",
    "commands": [
      "configure terminal",
      "ip route 2.2.2.0 255.255.255.252 192.168.1.2"
    ]
  }
}
```

* *Response*:

```
{
  "results": [
    { "command": "configure terminal", "status": "success" },
    {
    "command": "ip route 2.2.2.0 255.255.255.252 192.168.1.2",
    "status": "success" }
  ]
}
```

* *Description*: Applies a sequence of CLI commands to the specified device. Returns per-command status and any error messages.

4. *execute-cmd*

* *Request*:

```
{
  "method": "execute-cmd",
  "params": {
    "device": "R1",
    "command": "show ip route 2.2.2.0 255.255.255.252"
  }
}
```

* *Response*:

```
{
  "output": "S 2.2.2.0/30 [1/0] via 192.168.1.2"
}
```

* *Description*: Executes a read-only command on the specified device and returns its output. Must not alter device state.

4.3. Task Evaluation Interface

After the agent signals completion, the framework uses the Task Evaluation Interface to retrieve results:

* *export-final-cfg*

  - *Request*:

```
{
  "method": "export-final-cfg"
}
```

  - *Response*:

```
{
  "configs": {
    "R1": "!\nversion 15.2\n...",
    "R2": "!\nversion 15.2\n..."
  }
}
```

  - *Description*: Returns the final running-configuration of each device.

* *run-testcases*

  - *Request*:

```
{
  "method": "run-testcases",
  "params": {
    "testcases": [
      {
        "device": "R1",
        "commands": ["show ip route 2.2.2.0 255.255.255.252"],
        "expected_output": "S 2.2.2.0/30 [1/0] via 192.168.1.2"
      },
      ...
    ]
  }
}
```

- *Response*:

```
{
  "results": [
    {
      "name": "Verify primary static route on R1",
      "status": "pass"
    },
    {
      "name": "Verify backup static route on R1",
      "status": "fail"
    }
  ]
}
```

- *Description*: Executes each verification command sequence on the appropriate device and compares actual output against expected_output (regular expression). Returns pass/fail for each testcase.

5. Security Considerations

LLM-driven network configuration introduces risks such as unintended or malicious commands, emulator vulnerabilities, and data exposure; to mitigate these, NetConfBench should enforce strict input validation (e.g., YANG/XML schema checks), run emulated devices in isolated sandboxes with limited privileges, encrypt and restrict access to task definitions and logs, employ human-in-the-loop approval for generated configurations, and use curated prompt templates and fine-tuning to reduce LLM hallucinations.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/rfc/rfc6241>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/rfc/rfc7950>.

7.2. Informative References

[A2023] Hat, R., "Ansible", 2023.

[Fuad2024] Fuad, A., Ahmed, A. H., Riegler, M. A., and T. Cicic, "An intent-based networks framework based on large language models", 2024.

[Kreutz2014] Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and S. Uhlig, "Software-defined networking: A comprehensive survey", 2014.

[Lira2024] Lira, O. G., Caicedo, O. M., and N. L. S. da. Fonseca, "Large language models for zero touch network configuration management", 2024.

[Liu2024] Liu, C., Xie, X., Zhang, X., and Y. Cui, "Large language models for networking: Workflow, advances and challenges", 2024.

[Long2025] Long, S., Tan, J., Mao, B., Tang, F., Li, Y., Zhao, M., and N. Kato, "A Survey on Intelligent Network Operations and Performance Optimization Based on Large Language Models", 2025.

[Wang2024NetConfEval] Wang, C., Scazzariello, M., Farshin, A., Ferlin, S., Kostic, D., and M. Chiesa, "Netconfeval: Can llms facilitate network configuration?", 2024.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Yong Cui
Tsinghua University
Beijing, 100084
China
Email: cuiyong@tsinghua.edu.cn
URI: http://www.cuiyong.net/

Chang Liu
Tsinghua University
Beijing, 100084
China
Email: liuchang23@mails.tsinghua.edu.cn

Xiaohui Xie
Tsinghua University
Beijing, 100084
China
Email: xiexiaohui@tsinghua.edu.cn

Chenguang Du
Zhongguancun Laboratory
Beijing, 100094
China
Email: ducg@zgclab.edu.cn