Automatic Generation of SDN Intents from Natural Language Commands Using LLMs

Kejsi Bushi, Genti Haveri, Noah Bartocci, Manisha Kaushal

November 17, 2025

Contents

1 Introduction

Software Defined Networking has shaped the modern network management by dividing the control plane from the data one, and ensuring centralized control over the network behaviors. Now operators can define high level policies (also known as intents) that describe the network outcome. Before, the configuration of individual devices was done manually. While doing this, Large Language Models offer simplification of network configuration through natural language commands. But this combination has raised a security vulnerability. LLMS generate reasonable but also incorrect outputs. This is known as hallucination. When we apply this in network configuration, LLMs generate policies that may appear correct but at the same time fail to execute properly.

The latest research shows that fewer than 12percent of GPT-4- network configurations are executable without verification. This can place organizations at critical risk.

This topic matters a lot in cybersecurity. You cannot secure something that you cannot trust. The collaboration of AI driven automation and network security requires and demands strict verification mechanisms. Beyond hallucination, this research addresses semantic ambiguity in policy interpretation, conflict detection across multi-tenant environments, compliance validation, and real-time performance degradation under load—all of which have direct security implications.

The main question is: How can we create credible LLM based SDN intent generation that can prevent hallucinated policies from reaching production systems? Our research shows that the answer lies in the implementation of automated behavioral testing inside isolated sandbox environments, combined with feedback driven refinement. The best solution is to shift the question from "Can AI do this?" to "Can AI do this correctly with our verification?".

2 The Problem

Implementing automatic generation of Software-Defined Networking(SDN) intents using Large Language Models presents several technical, operational, and architectural challenges. Although there are some theoretical frameworks that show promise, practical deployment faces significant challenges related to the model reliability, system integration, performance constraints, and compliance requirements.In this section, every challenge will be discussed in detail, providing a clearer understanding of the key issues that impact real world adoption.

2.1 LLM Hallucination and Factual Correctness

LLM hallucination is the most critical implementation challenge for network configuration generation. These hallucinations occur when the modal generates network configuration that look correct at first, but are semantically wrong or executionally impossible.[11]. Research shows that fewer than 12% of GPT 4-generated network configurations are executable without formal constraint checking and verification. This statistic shows that direct LLM generation produces approximately 88% of unusable output when deployed on real network infrastructure. The configurations may parse correctly as JSON or YAML but it may violate network topology constraints, reference unavailable resources or create inconsistent rule sets.[8]

2.1.1 Manifestations in Network Configuration

Hallucinations in network contexts take several forms:
Phantom resource references: LLMs generate configurations that references switches, ports VLANs, or IP addresses that don't exist in the target network. Since these LLMs don't have access to real-time network inventory, they tend to invent components or reuse outdated resources, which leads to configurations that look valid syntactically but are impossible to deploy operationally.

Topology violations: Since the LLMs don't have visibility into the actual network topology, it does not know which devices are connected, how they are linked, or whether those links are active. This results in creating paths that may look reasonable in language but it does not exist in a real network.

Protocol mismatches: Configurations may specify protocols that are unsupported by target devices. For example, generating IPv6 rules for legacy switches lacking IPv6 support causes installation failures, misleading operators about deployment status.

Mathematical reasoning failures: One problem that LLMs still struggle with is mathematical and logical reasoning. For creating SDN intents, LLMs have to solve complex QoS calculations, bandwidth allocation arithmetic, or latency computations, tasks that require numerical consistency. What LLMs do in this case is approximate numbers or skip intermediate steps, producing intents that fail to satisfy required performance guarantees.

Syntactic correctness masking semantic errors: LLM-generated configurations can pass JSON or YANG schema but implement completely wrong network behavior. Eventhough that these LLMS can satisfy structural requirements, it does not guarantee that the configuration's intent or operational logic is correct.

2.1.2 Performance Degradation with Complexity

Experimental evaluation reveals accuracy degradation strongly correlated with rule complexity. Anouar El Hachimi at Politecnico di Milano discovered that these LLMs achieved up to 96.7% accuracy on simple flow rules. However, when the test cases become more challenging the accuracy decreased to approximately 73.3%. [3]. This complexity dependent failure rate can create unpredictable system behavior where simple intents deploy reliably while sophisticated policies fail frequently.

2.1.3 Hallucination Evolution Over Time

After extended operation periods, LLM-based systems exhibit signs of hallucination drift. As models repeatedly generate, refine or self - correct configurations, small inconsistencies can accumulate, diverging from the actual network state. This temporal degradation can be due to context window pollution, prompt injection from previous interactions or cumulative errors in few-show learning examples.[9]

2.1.4 Mitigation Challenges

There are different techniques that aim to reduce hallucinations like Retrieval-Augmented Generation(RAG), formal verification, etc, but still they cannot eliminate them entirely. For example RAG without domain specific retrieval drops accuracy to 46.7% in ablation studies.[3]

2.2 Semantic Ambiguity and Intent Interpretation

2.2.1 The Semantic Gap Problem

Natural language contains inherent ambiguity that creates severe challenges for intent-based networking. The semantic gap between high-level human expressions and low-level network specifications requires systems to infer missing context, resolve vague terminology and interpret user intent that is often under specified.

A simple example that we could show is the case when the user ask LLMs to "increase bandwidth". This command can mean several operations including:

- Allocate more physical bandwidth resources to a link.
- Reduce packet loss to improve effective throughput.
- Enable faster transmission rates through protocol optimization.
- Add redundant paths for load balancing.
- Upgrade physical media.
- Adjust QoS policies to prioritize traffic.

If the user doesn’t specify the prompt, the the LLMs make arbitrary choices based on training data biases rather than actual operator intent. These choices may contradict with what the user actually wanted, leading to security vulnerabilities or performance degradation.

2.2.2 Telecom-Specific Semantic Challenges

Generic LLMs trained on general purpose corpora lack deep understanding of telecommunications terminology, 3GPP specifications, and vendor specific jargon. An evaluation on telecom benchmarked showed that generic models achieve accuracy scores below 0.07 (7%) [5] on nuanced telecom understanding tasks. These models can indeed recognize keywords but fail to grasp semantic relationships specific to network technologies. Still there exists some models that are trained specifically on telecom vocabulary and have achieved accuracy scores exceeding 93% which is more than 13x improvement over general purpose models. However developing this specialized models, requires domain specific datasets by subject matter experts, representing significant investment.[5]

2.2.3 Paraphrase Sensitivity and Semantic Consistency

LLMs exhibit inconsistent responses to semantically equivalent paraphrased inputs. The same intent expressed differently may generate different configurations, revealing lack of true semantic understanding. Knowing this, it makes the system unreliable, as minor wording variations produce unpredictable results.

2.3 Validation, Conflict Detection and Assurance Challenges

Intent-based networking introduces complex conflict scenarios like:

- Intent to Intent Conflicts: Multiple users or application can request incompatible objectives. This is a massive problem in muti tenant environments where different tenants may request exclusive use of scarce resources or conflicting security policies. If an LLM is used for translation and enforcement it may synthesize two incompatible sets of SDN rules. This can result even in attacks where malicious users could craft conflicting requests deliberately, exploiting LLM prompt context to subvert network policies.
- Intent to Network Conflicts: These conflicts occur when the generated policies are incompatible with the actual capabilities or conditions of the network. A simple example can be when the user requests 10 Gbps throughput on a link that supports only 1 Gbps. [2]
- Policy to Policy Conflicts: Low level translated policies contradict each other even when high level intents might appear compatible. In SDN and intent-based networking each user intent is translated by a controller or LLM into precise policies like flow table entries, firewall rules or routing policies. In cases where these rules overlap in their match conditions, priorities or actions they can interfere with one another, which can result in packet drops, or even inconsistent security enforcement.[2]

2.4 Real-Time Performance and Latency Requirements

2.4.1 Latency Incompatibility

Network operations often demand sub-millisecond response times for control plane operations, meanwhile LLM interface introduces latencies of 100-500+ milliseconds for single generation requests. This mismatch creates fundamental tension between LLM capabilities and network requirements.
A major bottleneck is Time-to-First-Token which has 2 phases the prefill and decode phase.

In the prefill phase the entire input must be processed before the first token can be produced. This is an operation that can be computationally expensive but also is responsible for most of the delay which are unacceptable for real time decision making. Furthermore under high request loads, LLM system suffer queueing delays and throughput degradation. Experimental evaluation demonstrates that as request rates increase, average latency per token rises rapidly due to computational resource competition.[12]. Although there have been efforts to fix this problem, it still faces capacity limits.

2.5 Regulatory Compliance and Security Validation

LLMs does not posses any understanding of regulatory requirements, industry standards or compliance frameworks. Network configurations must comply with different regulations, including Government regulations, industry standards, technical specifications and organizational security policies. [4]This creates a significant risk that LLM generated outputs may violate mandatory rules. Furthermore LLMs cannot verify whether a configuration satisfies security controls. This makes it necessarily the creation of explicit validation mechanisms in order to make sure that every generated configuration undergoes regulatory compliance checks.[10]

2.6 Scope Limitations and Feasibility Constraints

Although this section identifies a wide range of challenges associated with LLM-based intent generation, we should acknowledge that its difficult to develop a complete solution for all of the problems mentioned. Challenges that are mentioned remain only partially addressed and organizations are still exploring methods to reduce hallucinations, detect policy conflicts and ensure that everything meets regulatory requirements.

2.7 Existing IBN Systems and Their Validation Mechanisms

Existing IBN systems implement comprehensive validation frameworks spanning the entire intent lifecycle: intent profiling, translation, resolution activation and assurance. In this section we will analyze current IBN implementations and how they address the problems mentioned in the above section through conflict detection, continuous monitoring that any LLM based approach must distinguish itself from.

2.7.1 Commercial IBN Systems

Juniper Apstra is a system that provides Policy Assurance with automatic conflict detection when policies overlap, real-time evaluation of deployed intents and drift detection between intended and actual network state. Intent Based Analytics(IBA) continuously validates the network while a Root Cause Identification System performs anomaly analysis when violations occur.[7]
Cisco DNA center employs continuous assurance using machine learning to predict issues before they manifest. What this platform does is that in cases where it finds deviations from intended state is automatically takes corrective actions.

2.7.2 Emerging LLM Integrated IBN Systems

Recent research has introduced platforms that use large language models to automate intent translation, conflict detection and validation in network environments. These systems goal is to make intent specification more natural and responsive but at the same time they face some reliability and validation challenges.

- NetIntent(ODL and ONOS controllers): This system evaluates the ability of Large Language Models to support the complete IBN life cycle on industry grade SDN controllers like OpenDaylight and ONOS. Using benchmarks like Intent2Flow and FlowConflict, the framework assesses 33 LLMs across multiple IBN tasks. However there were some problems identified while experimenting. LLMs show drastically different results where some models excel at the intent translations while others fail at conflict detection.[6]
- IntentLLM(TeraFlowSDN): It integrates a LLM chatbot into the TeraFlowSDN controller. This chatbot enables operators to create, query and explain network intents using natural language. After the LLM chatbot receives the user request, it tries to find existing intents, while explaining the purpose of network configurations, and then generate new intents and commands for deployment in TeraFlowSDN. Although it has many strengths, like the fact that everything is done via conversational chat and that its capable to explain every concept, still has its own problems. LLM hallucination is their biggest downside, requiring human confirmation or stronger post processing before operator confirmation and deployment. Furthermore, the intents created, could be incomplete which may result into causing adverse network behavior or service outages.[1]

3 The Goal

Large language models (LLM) have improved a lot during the recent years, recently they have been used to translate natural-language network policies into SDN intents, but they usually generate these intents based on either patterns from their training data or without any relation to the user's specific network environment. In practice this leads to the LLM proposing intents, that appear correct but often don't work in the applied network either because it is not compatible with the user's environment or because the intent itself has logic flaws, for example, referring to non-existent devices or violating the current network states. Prior studies mostly focused on the intent translation but they lack the validation steps like conflict detection and environment-specific checks. Without this mechanism to verify if the LLM's output is actually feasible or deployable in the users environment, these AI-generated intents can fail during deployment or even introduce misconfigurations, since the model cannot confirm if the intents it generated are correct for the targeted SDN controller and topology.

The goal of our research is to propose a solution for this problem and integrate an automated validation loop into the LLM-based intent generation process. Instead of directly trusting the LLM's output, our solution would first test the generated intent in a sandbox SDN environment (a Mininet-emulated network running ONOS controller) before given the result to the user. The LLM's natural language output will first be translated into an intent (JSON or flow rule), then it will be deployed in the simulated network to observe its behaviour and sure that it achieves user's intended outcome. This closed-loop verification allows the system to catch and correct any error or conflicts (for example: flow rules that cannot be installed or flow rules that cause unintended traffic behaviour) and directly generate the needed solutions. In conclusion, the goal is to enhance the trustworthiness and accuracy of LLM-derived network intents. The user will only receive intents that are proven to work in a realistic lab setting which reduces the risk of failures or misconfigurations. The next section will focus on our solution to realize this LLM-based intent generation system with a sandbox testing mechanism.

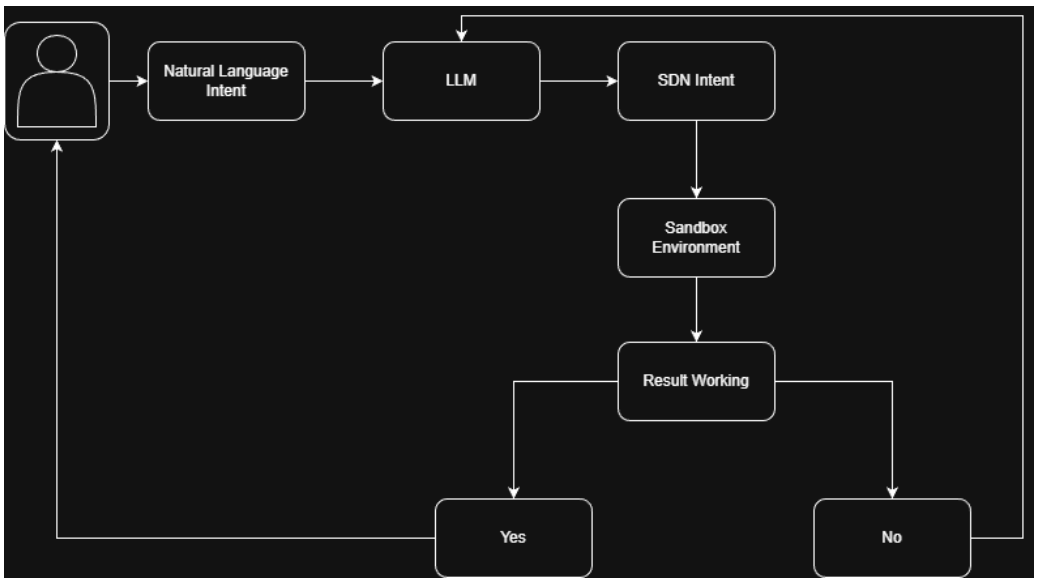

Figure 1: Setup

4 The Solution

To address the challenges outlined in Section 2, we propose a verification-driven framework that combines Large Language Models (LLMs) with automated testing in a sandboxed Software-Defined Networking (SDN) environment. Instead of trusting LLM-generated configurations directly, our solution always tests them first. Only configurations that pass functional tests are considered correct. This creates a closed-loop validation process where intents are translated, tested, and refined before they are accepted.

4.1 System Architecture

Our solution architecture consists of five integrated components that work together in a feedback loop:

1. **Natural Language Interface**: The user can describe what they want in simple sentences (for example, "Block traffic from h1 to h2"), without needing to know network commands or syntax.
2. **LLM Translation Engine**: This component sends the user's intent (and optionally network context) to a commercial LLM API (Groq Llama-3.3-70B) and receives a candidate configuration in return (for example, an iptables rule or a structured JSON object).
3. **Sandbox Testing Environment**: A Mininet-emulated network where the candidate configuration can be applied and tested safely. This sandbox behaves like a small real network, but does not affect any production system.
4. **Automated Verification Module**: This module runs connectivity tests (such as ping) before and after applying the configuration. It checks whether the network behaviour matches what the user asked for.
5. **Intelligent Feedback Loop**: If the configuration does not achieve the intended effect, this component creates clear feedback (for example, "Traffic still flowing, 0% packet loss") and sends it back to the LLM, so that the model can try a different configuration.

This architecture ensures that we never rely on LLM output. Every configuration is functionally tested in the sandbox. This directly addresses the reliability and hallucination problems discussed in Section 2.

```
mininet@mininet-vm:~/llm_validation_system$ sudo -E python3 working_llm_syst
em.py
======================================================================
 REAL LLM-BASED SDN INTENT GENERATOR
======================================================================

[Architecture]
 LLM:          Groq (Llama-3.3-70B)
 Network:      Mininet
 Verification: Automated testing
 Feedback:     Closed-loop refinement

[Press Enter to start...]
```

Figure 2: System architecture: user intent, LLM translation (Groq Llama-3.3-70B), Mininet sandbox, automated verification, and feedback loop

4.2 Implementation Components

4.2.1 LLM Integration Module

We integrated Groq's Llama-3.3-70B model through authenticated REST API calls to translate natural-language intents into network-level outputs. All API credentials were stored securely as environment variables and never hardcoded into the system. The LLM integration module:

- Builds structured prompts that include the user intent and, when needed, a short description of the network topology (hosts, IP addresses, and switch).
- Sends authenticated API requests to Groq and retrieves the model's responses in real time.
- Parses the responses, which can be raw commands (such as iptables rules) or structured JSON objects that describe which traffic should be allowed or dropped.
- Uses verification feedback (for example, "0% packet loss, rule did not work") to ask the LLM for refined configurations in follow-up calls.

The prompts clearly specify the desired format of the output, the available hosts and IP addressing scheme, and the technologies in use (iptables, OpenFlow). This reduces ambiguity and helps generate more structured and machine-usable outputs.

4.2.2 Network Emulation Layer

We use Mininet as the testing environment. The emulated topology includes several virtual hosts (h1, h2, h3) in the 10.0.0.0/24 subnet, connected through OpenFlow-enabled virtual switches. Mininet supports real network protocols, so standard tools, such as `ping` , can be used to test connectivity and measure packet loss.

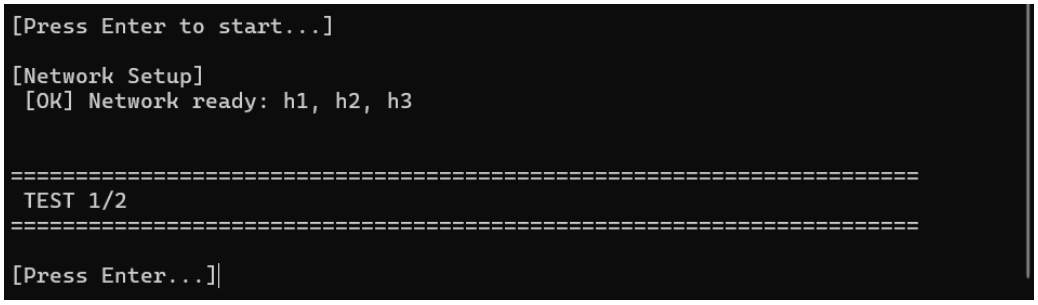

Figure 3: Mininet network setup with three hosts (h1, h2, h3) connected to a switch and ready for configuration testing

4.2.3 Verification and Testing Module

The verification module performs simple but effective behavioural tests:

- **Baseline Measurement**: Before applying any configuration, the system tests connectivity (for example, `h1` pinging `h2`) to confirm normal behaviour (typically 0% packet loss).
- **Configuration Deployment**: The LLM-generated command is applied inside the Mininet environment (for example, an iptables rule on a host).

- **Post-Deployment Verification**: The same connectivity test is repeated to see whether behaviour has changed (for example, has the ping started failing?).
- **Intent Validation**: The system compares the observed behaviour with the user's intent. If traffic was supposed to be blocked, a successful configuration should change packet loss from 0% to 100%.

4.3 Verification Workflow

Putting these components together, our workflow from natural language to verified configuration is:

1. **Intent Reception**: The user provides a natural language request (for example, "Block traffic from h1 to h2").
2. **LLM Translation**: The intent (and optionally topology context) is sent to Groq's LLM, which generates a candidate configuration (for example, an iptables rule or a JSON rule object).
3. **Baseline Testing**: The system measures current connectivity to establish a reference point before any change.
4. **Sandbox Deployment**: The candidate configuration is applied to the Mininet environment.
5. **Functional Verification**: Automated tests (such as repeated pings) observe the network behaviour after deployment.
6. **Intent Validation**: The system checks if the new behaviour matches the intent (for example, if traffic is now blocked).
7. **Outcome Decision**:
    - **Success**: The configuration is marked as verified for this intent.
    - **Failure**: The system records that the intent was not achieved and generates feedback for the LLM, triggering another refinement attempt.

4.4 Live System Demonstration

We built a working prototype that connects all these pieces. The system sends real natural language intents to Groq's LLM, receives configurations in response, applies them in a Mininet network, and then verifies whether they behave as intended.

4.4.1 Experiment 1: Traffic Blocking Intent with iptables

User Intent: "Block traffic from h1 to h2"

In the first experiment, the system used the LLM to generate iptables commands to block traffic from host `h1` (10.0.0.1) to host `h2` (10.0.0.2). Across three attempts, the LLM produced the following commands:

```
Attempt 1: iptables -I OUTPUT -d 10.0.0.2 -j DROP
Attempt 2: iptables -I FORWARD -s 10.0.0.1 -d 10.0.0.2 -j DROP
Attempt 3: iptables -I FORWARD -s 10.0.0.1 -d 10.0.0.2 -j DROP
```

Baseline testing showed 0% packet loss between `h1` and `h2`, as expected in a normal network. After applying each configuration, we repeated the ping tests. In all three attempts, packet loss remained 0%, meaning that traffic was not blocked and the rules had no effect on actual connectivity.

```
================================================================
 USER INTENT: Block traffic from h1 to h2
================================================================

--- Attempt 1/3 ---

[1/5] LLM Translation
      [LLM] Querying Groq API...
      [LLM] Generated: iptables -I OUTPUT -d 10.0.0.2 -j DROP

[2/5] Baseline Test
      Before: 0% loss (REACHABLE)

[3/5] Deploy Configuration
      Applied: iptables -I OUTPUT -d 10.0.0.2 -j DROP

[4/5] Verification Test
      After: 0% loss (REACHABLE)

[5/5] Validation
      [FAIL] Intent not satisfied. Expected fully blocked, but got 0% loss.

--- Attempt 2/3 ---

[1/5] LLM Translation
      [LLM] Querying Groq API...
      [LLM] Generated: iptables -I OUTPUT -d 10.0.0.2 -j REJECT --reject-with icmp-host-unreachable

[2/5] Baseline Test
      Before: 0% loss (REACHABLE)

[3/5] Deploy Configuration
      Applied: iptables -I OUTPUT -d 10.0.0.2 -j REJECT --reject-with icmp-host-unreachable

[4/5] Verification Test
      After: 0% loss (REACHABLE)

[5/5] Validation
      [FAIL] Intent not satisfied. Expected fully blocked, but got 0% loss.
```

Figure 4: First two generation attempts: LLM output and verification. Both attempts fail to block traffic despite appearing syntactically correct.

```
--- Attempt 3/3 ---

[1/5] LLM Translation
      [LLM] Querying Groq API...
      [LLM] Generated: iptables -I OUTPUT -d 10.0.0.2 -j REJECT --reject-with icmp-host-unreachable

[2/5] Baseline Test
      Before: 0% loss (REACHABLE)

[3/5] Deploy Configuration
      Applied: iptables -I OUTPUT -d 10.0.0.2 -j REJECT --reject-with icmp-host-unreachable

[4/5] Verification Test
      After: 0% loss (REACHABLE)

[5/5] Validation
      [FAIL] Intent not satisfied. Expected fully blocked, but got 0% loss.

[FAIL] Could not verify intent after all attempts.
```

Figure 5: Third attempt with refined iptables rule. Verification again shows 0% packet loss, confirming failure to enforce the intent.

Verification Result: All three attempts failed. The configuration was applied, but traffic was still flowing normally.

```
================================================================
 FINAL RESULTS
================================================================

 1. [FAIL] Block traffic from h1 to h2
    Attempts: 3

 Success Rate: 0%

================================================================
 KEY DEMONSTRATION ELEMENTS
================================================================
 [OK] Real LLM called via Groq API
 [OK] Natural-language intent processed
 [OK] Network behavior validated in Mininet
 [OK] Closed-loop verification workflow executed

 Experiment finished.
```

Example Feedback: “Expected traffic to be blocked, but measured 0% packet loss. Command did not change behaviour. Traffic still flowing.”

This experiment supports our main claim: LLMs can generate commands that look correct but do not work in practice, especially when they are unaware of how traffic actually flows in the emulated network.

4.4.2 Experiment 2: Topology-Aware Structured Translation

In the second experiment, we focused on how the LLM represents the intent when it is given a clear view of the network. Instead of asking for a ready-to-run command, we provided a simple topology description (hosts, IPs, and switch) and asked the LLM to output a structured JSON rule that describes which traffic should be dropped.

Input: Topology summary + user intent (“Block traffic from h1 to h2”)

Example LLM Output:

```
{
  "switch_id": "s1",
  "match_src_ip": "10.0.0.1",
  "match_dst_ip": "10.0.0.2",
  "action": "drop"
}
```

Figure 6: Structured JSON rule generated by the LLM when given topology context and a blocking intent

This output is easy for a Python script to parse and can be converted into an OpenFlow rule or an iptables command in a controlled way. Experiment 2 does not test enforcement directly, but it shows that the LLM can produce clean, machine-readable policies when it has access to simple topology information and when the output format is clearly specified.

4.4.3 Summary of Demonstration Results

Across these experiments, we observed the following:

- LLMs can generate network commands that look correct but do not change real traffic behaviour (Experiment 1).
- Automated verification in Mininet successfully detects these failures by comparing pre- and post-deployment connectivity.
- When provided with explicit topology context and a required output format, the LLM can produce well-structured JSON rules that are easier to validate and enforce programmatically (Experiment 2).

4.5 Demonstration Significance

The outcomes of the live system demonstration are important for two reasons. First, they show that the problem is real: even for a very simple intent like blocking traffic between two hosts, the LLM's suggested configurations can completely fail in practice. Second, they show that the solution is necessary: without automated verification, these non-functional configurations could be mistakenly trusted and deployed.

Our prototype confirms that functional testing in a sandbox can reliably distinguish between configurations that only look right and configurations that truly implement the intended behaviour.

4.6 Addressing Identified Challenges

The solution directly addresses the challenges from Section 2:

4.6.1 Mitigating LLM Hallucinations

The experiments demonstrate that the LLM sometimes produces plausible but non-functional network commands. Our verification framework:

- Detects non-functional configurations based on real network behaviour.
- Prevents hallucination or incorrect policies from being accepted.
- Provides clear feedback messages that can be used to refine future LLM generations.

4.6.2 Resolving Semantic Ambiguity

Observing actual traffic, the system checks what the network really does instead of only trusting the written configuration. This helps catch cases where the LLM misunderstood the intent or where the same words could be interpreted differently. Functional tests make the interpretation explicit.

4.6.3 Intent-to-Network Conflict Detection

If the network cannot implement a requested intent because of constraints, missing capabilities, or policy conflicts, the verification step will reveal that there is no observable behavioural change. This is safer than deploying untested rules and discovering such conflicts directly in a production network.

4.7 System Advantages

Our verification-driven approach offers several practical benefits:

1. **User Accessibility**: Users can express their needs in natural language without learning the syntax of network commands.
2. **Deployment Safety**: All testing happens in an isolated Mininet sandbox, protecting production networks from misconfigurations.
3. **Configuration Reliability**: Only configurations that pass behavioural tests are considered correct, which reduces the "looks valid but does not work" problem.
4. **Adaptive Improvement**: Feedback from failed attempts can be used to guide the LLM toward better configurations over time.
5. **Transparency**: Users can see test results (for example, packet loss before and after) as evidence that the configuration has been checked.

5 Future Work and Possible Extensions

1. Ghost Resource Prevention Layer

A future extension is to add a validation layer that rejects any LLM-generated rule referencing non-existent hosts, invalid switch ports, or unsupported protocols. This prevents "ghost configurations" that are syntactically correct but impossible to enforce.

2. Network State Snapshot

Before generating rules, the system could provide the LLM with a live snapshot of the current network state, including host mappings, ARP tables, link status, and controller flow entries. Supplying this context would help the model produce more accurate, topology-aware configurations.

3. Self-Healing Refinement Loop

The system can be extended with automatic self-healing logic that corrects syntax errors, switches between enforcement mechanisms (iptables, OpenFlow, ACLs), and tests alternative blocking strategies after each failed attempt. This would make the pipeline more adaptive and resilient.

4. Human-in-the-Loop Approval

Another potential improvement is to allow administrators to review, modify, or approve LLM-generated rules before deployment. This step ensures oversight for sensitive configurations and adds a safe manual control layer.

5. Rule-Based Compliance Checker

A compliance module could automatically validate that generated configurations follow organizational policies, GDPR data-flow constraints, zero-trust security principles, allowed port ranges, and domain-specific access control rules. This would ensure that all accepted configurations remain consistent with regulatory and security requirements.

6 Conclusion

Our solution demonstrates the feasibility and necessity of verification-driven LLM-based network configuration generation. The live demonstration with real LLM integration empirically validates that:

1. Large Language Models generate unreliable network configurations requiring validation, even for straightforward intents
2. Automated functional testing effectively detects configuration failures that syntactic validation would miss
3. Feedback-driven iterative refinement enables LLMs to adapt and attempt alternative approaches
4. The verification framework prevents deployment of non-functional configurations, addressing a critical gap in current LLM-based automation approaches

By combining natural language understanding with rigorous automated testing, we achieve a system that is both accessible to non-expert users and reliable through systematic validation. The key innovation lies not in LLM capabilities alone, but in the verification framework that ensures generated configurations are functionally tested before any deployment consideration.

This work establishes a foundation for trustworthy AI-driven network automation, demonstrating that with appropriate validation mechanisms, LLM-based intent generation can move from research prototype toward practical deployment consideration. The demonstration validates our core contribution: automated verification is essential for reliable LLM-based network configuration generation.

7 AI Assistance Disclosure

We have used ChatGPT and Perplexity as a reference and learning tool to understand the concepts, gather insights, and cross-check information from credible sources. It assisted us in framing the responses, but the final content reflects our understanding.

References

[1] Daniel Adanza et al. “IntentLLM: An AI Chatbot to Create, Find, and Explain Slice Intents in TeraFlowSDN”. In: *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. 2024, pp. 307–309. DOI: 10.1109/NetSoft60951.2024.10588917.

[2] Wenlong Ding et al. *Automating Conflict-Aware ACL Configurations with Natural Language Intents*. arXiv preprint arXiv:2508.17990. 2025. URL: https://arxiv.org/abs/2508.17990.

[3] Anouar El Hachimi et al. *Flow–Rule Generation for SDN Using LLMs with Retry-Based Deployment Validation*. Slides presented at *IETF 124, Network Management Research Group (NMRG)*. Available at: https://datatracker.ietf.org/meeting/124/materials/slides-124-nmrg-sessa-flow-rule-generation-for-sdn-using-llms-with-retry-based-deployment-validation-00. 2025.

[4] Inc. Gluware. *How Network Automation Ensures Compliance and Reduces Security Risks*. Blog article, Gluware. Aug. 2021. URL: https://gluware.com/automate-compliance-blog/.

[5] GSMA. *Closing the AI Accuracy Gap in Telecoms: The Critical Role of Domain-Specific Language Models*. Newsroom article, GSMA. May 2025. URL: https://www.gsma.com/newsroom/article/closing-the-ai-accuracy-gap-in-telecoms-the-critical-role-of-domain-specific-language-models/.

[6] Md Hossain and Walid Aljoby. "NetIntent: Leveraging Large Language Models for End-to-End Intent-Based SDN Automation". In: *2025* (2025).

[7] Inc. Juniper Networks. *Apstra Policy Assurance: Quick Start Guide*. Tech. rep. Available online at https://www.juniper.net/documentation/us/en/software/apstra5.0/apstra-policy-assurance/apstra-policy-assurance.pdf. Juniper Networks, Inc., 2024.

[8] Subbarao Kambhampati et al. "Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks". In: *Proceedings of the Forty-first International Conference on Machine Learning*. 2024.

[9] Dimitrios Michael Manias, Ali Chouman, and Abdallah Shami. *Semantic Routing for Enhanced Performance of LLM-Assisted Intent-Based 5G Core Network Management and Orchestration*. arXiv preprint arXiv:2404.15869. 2024. URL: https://arxiv.org/abs/2404.15869.

[10] Selector AI. *Network Automation in 2025: Technologies, Challenges, and Solutions*. Learning Center article, Selector AI. 2025. URL: https://www.selector.ai/learning-center/network-automation-in-2025-technologies-challenges-and-solutions/.

[11] Huzaifa Sidhpurwala. *When LLMs day dream: Hallucinations and how to prevent them*. 3-minute read. Red Hat. Sept. 2024. URL: https://www.redhat.com/en/blog/when-llms-day-dream-hallucinations-how-prevent-them.

[12] Jin Yang et al. *Quality-of-Service Aware LLM Routing for Edge Computing with Multiple Experts*. arXiv preprint arXiv:2508.00234. 2025. URL: https://arxiv.org/abs/2508.00234.