Net2d-LLM: Translating Structured Network Intents into CLI using LLMs with Execution in a Network Digital Twin

Jerônimo Menezes[1], **Leonardo Bitzki**[1], **Diego Kreutz**[1]

[1]AI Horizon Labs
Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (UNIPAMPA)

jscmenezes@gmail.com, bitzki@ufrgs.br, diegokreutz@unipampa.edu.br

Abstract. *This paper evaluates the use of Large Language Models (LLMs) to translate structured network intents, expressed as Desired State Models (DSMs), into vendor–specific CLI configurations. Net2d–LLM implements a deterministic, single–pass pipeline in which DSMs are rendered into constrained prompts and translated into CLI commands executed within a Network Digital Twin (NDT). The evaluation compares multiple LLMs under identical conditions, using latency, token usage, efficiency, and configuration consistency as assessment dimensions. The results show that general–purpose LLMs, when guided by structured prompts, can reliably generate syntactically valid and functionally correct configurations, demonstrating reproducibility and confirming the feasibility of LLM–driven automation in multivendor network environments.*

1. Introduction

Configuring network devices remains a critical and error-prone activity, particularly in heterogeneous environments with multiple vendors and software versions. Traditional automation techniques based on templates and scripts reduce operational effort but are brittle under contextual variations and require continual expert maintenance. In parallel, recent work has explored the use of Large Language Models (LLMs) to translate high-level intents into device configurations. However, most approaches rely on free-form natural language input, which introduces ambiguity and limits objective, reproducible evaluation [Mondal et al. , Wang et al.]. Recent surveys highlight the absence of standardized inputs, structured benchmarks, and consistent evaluation protocols in this area [Hong et al. , Liu et al.].

This work adopts *Desired State Models* (DSMs), that are structured JSON representations of the target configuration which may be originated from a Network Source of Truth (NSoT) or management systems. DSMs are translated into CLI using a single vendor-agnostic prompt template that constrains the LLM to emit concise, command-only output. The resulting CLI is executed in a *Network Digital Twin* (NDT) composed of virtualized devices, enabling measurement of latency, token usage, and token efficiency, as well as functional validation through data-plane connectivity.

The contributions of this paper are fourfold: (i) a structured and standardized set of DSMs, together with a reproducible evaluation protocol; (ii) a vendor-agnostic prompt template that guides deterministic CLI generation across multiple models; (iii) a quantitative comparison of several LLMs under identical conditions using metrics such as latency, token counts, and efficiency; and (iv) a fully reproducible workflow composed of logs, scripts, and consolidated `CSV` outputs, enabling further analysis and community reuse.

The scope of this study focuses on Layer 2 operations, such as VLAN definitions and port modes, to minimize variability and maximize reproducibility. Extending the approach to broader configuration domains and multivendor environments is left as future work, supported by the same experimental methodology.

2. Related Work

The use of Large Language Models (LLMs) in network automation has progressed rapidly, particularly in translating high level intents into device configurations [Wei et al. 2025a, Mekrache et al. 2024, Angi et al. 2025, Mekrache and Ksentini 2024, Fuad et al. 2024, Tageldien et al. 2025, Tu et al. 2025, Wei et al. 2025b]. Most existing approaches rely on free form natural language, which introduces ambiguity, increases variability in generated outputs, and complicates reproducible evaluation. Recent surveys highlight the absence of standardized benchmarks, structured input formats, and consistent experimental protocols in this domain [Hong et al. , Liu et al.].

Existing work can be grouped into two categories. The first comprises natural language driven approaches such as NETBUDDY [Wang et al.], S Witch [Jeong et al.], LLM NetCFG [Lira et al.], and CoSynth VPP [Mondal et al.]. These systems translate textual descriptions into configurations or policies using prompting strategies, syntax checking, or simulated verification. Despite their expressiveness, they remain sensitive to linguistic variation and typically do not execute CLI on real or virtualized devices.

The second category includes structured model based approaches such as AppleSeed [Lin et al.] and the work by Hollósi et al. [Hollósi et al.], which rely on formal intents or YANG models to generate NETCONF configurations under schema constraints. These efforts improve determinism but do not explore the use of general purpose LLMs nor evaluate actual CLI execution.

Tabela 1. Summary of LLM-based network configuration approaches (2023–2025).

Work	Input Type	Output	Execution / Verification
NETBUDDY (2023)	Natural language (NL)	Conceptual config	No device execution
S-Witch (2024)	Natural language (NL)	CLI	Simulated NDT
LLM-NetCFG (2024)	Natural language (NL)	CLI	Dry-run only
AppleSeed (2023)	Structured intents	Multi-domain	Schema-based checks
Hollósi et al. (2024)	YANG models	NETCONF	Schema validation
CoSynth/VPP (2023)	NL + base config	CLI	External verifier
(Net2d-LLM)	**DSM (JSON)**	**CLI**	**Real NDT execution**

Table 1 summarizes representative efforts. None combine structured intents, deterministic prompt rendering, and real CLI execution in a Network Digital Twin (NDT). In contrast, our work adopts JSON based DSMs, a single vendor agnostic prompt template, and real execution in a virtualized network environment, enabling quantitative comparison across multiple LLMs under a fully reproducible protocol.

3. Net2d-LLM Architecture

Net2d-LLM implements a single-pass pipeline that translates structured intents into vendor-specific CLI and executes them on virtual devices in a Network Digital Twin (NDT). The pipeline fixes the input structure, prompt rendering, and execution environment so that different LLMs can be compared under identical conditions. Figure 1 summarizes the workflow: a Desired State Model (DSM) is rendered into a prompt, translated into CLI, and executed in the NDT, with logs and metrics recorded for analysis.

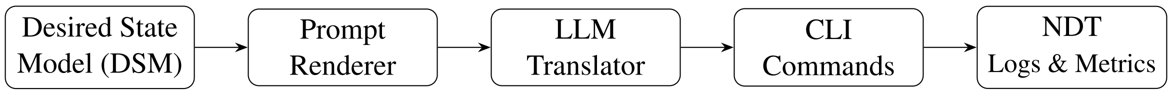

Figura 1. High-level Net2d-LLM pipeline.

3.1. Pipeline Overview

The pipeline has three stages. First, DSMs encoded in JSON specify interface parameters, VLANs, and operational mode. Using a single structured format ensures that all LLMs receive the same technical context.

Second, a vendor-agnostic prompt renderer converts the DSM into a constrained prompt that instructs the LLM to output only CLI commands, without prose or comments. The same prompt structure is used across models, and each LLM is invoked with fixed API parameters.

Third, the resulting CLI is executed in the NDT, and device output and token statistics are saved automatically. Correctness is assessed manually through inspection of logs and data-plane connectivity; no automated diffing or remediation loop is implemented. This keeps the pipeline simple and reproducible while still exercising real device behavior.

The NDT environment is intentionally minimal, serving solely as a functional validator for the CLI produced by the LLMs. Because the aim is to evaluate the translation pipeline rather than vendor-specific behavior or network design, focusing on Layer 2 primitives and a single virtualized platform keeps the environment simple, reproducible, and free of protocol-level variability. This choice isolates the contribution of the DSM$\rightarrow$prompt$\rightarrow$LLM$\rightarrow$CLI pipeline, which is the central focus of this work.

3.2. Desired State Models (DSMs)

A DSM is a structured representation of the desired configuration and may originate from a Network Source of Truth (NSoT) or other management systems. It captures intent in a machine readable format and removes the ambiguity typically associated with free form descriptions. Listing 1 shows a simplified example used in the evaluation, illustrating how operational parameters can be expressed in a consistent and vendor agnostic manner.

Using a single DSM structure for all models ensures that any differences in the generated CLI result from model behavior rather than variations in the input. This uniformity is essential for reproducible evaluation and allows direct comparison across LLMs under identical conditions. It also reinforces the separation between intent specification and configuration translation, a key requirement for deterministic and scalable network automation workflows.

Listing 1. Example Desired State Model (DSM).

```
{
  "device": "nexus01",
  "interface": {
    "name": "Ethernet1/2",
    "mode": "trunk",
    "tagged_vlans": [
      { "vid": 10, "name": "VLAN010" },
      { "vid": 20, "name": "VLAN020" }
    ],
    "untagged_vlan": { "vid": 20, "name": "VLAN020" }
  }
}
```

3.3. Prompt Renderer and LLM Translator

The prompt renderer applies a single deterministic template that injects device and interface information into a fixed instruction block. Constraints include producing only configuration commands, avoiding invented parameters, and ensuring that referenced VLANs are created before being applied. The complete template is provided in the experimental artifacts.

The LLM translator submits the rendered prompt to the selected model and records the raw output, latency, and token usage. No vendor-specific logic is embedded in the translator itself; vendor details appear only in the rendered prompt.

3.4. Execution in the NDT and Resulting Artifacts

CLI commands are applied to virtual devices (NX-OSv) using standard configuration-mode handling. All outputs, including the prompt, LLM response, extracted commands, device return, and usage metadata, are stored in per-run directories. These artifacts enable manual validation and support the quantitative analysis presented in Section 4.

3.5. Notes on Determinism

The pipeline encourages *syntactic determinism* (stable CLI shape) by fixing inputs and prompt structure, and *functional determinism* (equivalent operational state) through repeated execution under identical conditions. In this work, determinism is assessed qualitatively via log inspection and connectivity checks.

4. Experimental Evaluation

This section evaluates how different LLMs behave when translating structured intents (DSMs) into vendor-specific CLI using the single-pass pipeline described in Section 3.

Experiments were performed in a Network Digital Twin (NDT) composed of virtualized Cisco NX-OSv devices and Linux hosts. All executions were automated, and each stage of the pipeline produced log artifacts for inspection.

4.1. Objectives and Setup

The study investigates two questions: (i) whether LLMs generate syntactically valid and functionally correct CLI under a fixed input structure and prompt; and (ii) how latency, token usage, and efficiency vary across models.

The evaluation is strictly *single-pass*, with no automated remediation. Each run executes the sequence DSM $\rightarrow$ prompt $\rightarrow$ LLM $\rightarrow$ CLI $\rightarrow$ device, producing artifacts such as `prompt.txt`, `llm_response.txt`, `device_return.txt`, and `usage.json`. Functional correctness was checked by inspecting device outputs and verifying end-to-end connectivity (e.g., successful `ping` between `lab-ger` and `hostA`).

4.2. DSM Set and Execution Rounds

The evaluation used a small but representative set of structured DSMs covering common Layer 2 scenarios, including access ports, trunk ports with multiple tagged VLANs, and different choices of native VLAN. Each DSM was executed exactly once per model under identical conditions, reflecting the single-pass philosophy of this work. The goal is not to optimize or correct model responses through retries or feedback loops, but rather to observe each LLM's raw behavior when exposed to the same deterministic input. The complete DSM set used in the evaluation is included in the public repository.

4.3. Metrics

Four quantitative metrics are considered:

- **Average latency (ms)**: mean LLM response time;
- **Maximum latency (ms)**: worst-case response time;
- **Average and maximum tokens**: number of tokens generated per prompt;
- **Efficiency (tokens/s)**: total tokens divided by total time.

These metrics are objective and reproducible, as they are extracted directly from the recorded artifacts.

4.4. Results

Table 2 summarizes the results for the four evaluated models. Each model processed the same DSM set under identical conditions.

Tabela 2. Performance comparison across evaluated LLMs. Efficiency = tokens/s.

LLM	Avg. Lat. (ms)	Max Lat. (ms)	Avg. Tokens	Max Tokens	Efficiency
ChatGPT (gpt-4o)	2870	4999	585	602	203.83
DeepSeek (deepseek-chat)	3798	4884	653	691	171.92
Groq (gpt-oss-20b)	2156	2873	1184	1465	549.55
Gemini (gemini-2.5-flash)	6792	9685	1746	2201	257.23

Across DSMs, latency and token usage exhibited low variability within each model, with no failed executions or outlier behaviors. These observations suggest that the differences reported in Table 2 are primarily due to intrinsic model performance and API behavior, rather than instability in the pipeline itself.

All models produced syntactically valid and functionally correct CLI, achieving connectivity between `lab-ger` and `hostA`. Groq-20B exhibited the highest efficiency (549 tokens/s) and lowest average latency. GPT-4o offered the best balance between speed and consistency. Gemini 2.5 Flash and DeepSeek-Chat showed higher latency but maintained functional completeness.

4.5. Qualitative Observations

Inspection of logs showed that all models adhered to the prompt constraints. Differences were limited to command ordering and spacing, with no functional impact. No destructive or unexpected operations were generated, and all VLAN and interface configurations were applied successfully. Performance differences mainly reflect model architecture and API latency rather than inconsistencies in the pipeline.

4.6. Artifact Availability

To ensure full reproducibility, all DSMs, prompts, model outputs, device logs, and consolidated evaluation metrics will be made publicly available at:

`https://gitlab.com/net2d-community/net2d-labs`

The repository will also include the helper scripts used for data aggregation and analysis, enabling complete replication of the experimental workflow.

5. Conclusion and Future Work

This paper presented Net2d LLM, a single pass pipeline that translates structured network intents (DSMs) into vendor specific CLI using general purpose language models. By fixing the DSM format, prompt template, and execution environment, the approach enables reproducible comparison across models and reduces the ambiguity found in natural language inputs. Real device execution in an NDT allows direct assessment of syntactic and functional correctness.

All evaluated models produced valid and consistent configurations under a strict single pass workflow. Structured prompts stabilized model behavior, and each LLM exhibited a distinct performance profile: Groq 20B was the most efficient, GPT 4o offered the best balance between speed and stability, and Gemini 2.5 Flash and DeepSeek Chat maintained functional completeness despite higher latency. The evaluation, however, was limited to Layer 2 operations, a single network operating system, and manual inspection of logs.

Compared to prior work, Net2d LLM combines structured DSM inputs, deterministic prompt rendering, and execution in a network digital twin, creating a reproducible evaluation pipeline. Future work includes expanding to a multivendor NDT, integrating open source and locally hosted models, automating configuration verification, and publishing a larger DSM benchmark. All artifacts from this study will be released to support reproducibility and ongoing research.

Acknowledgements

This research received partial support from the RNP[1], through the IoTEdu[2] Working Group, and from the CAPES[3], under Financing Code 001.

Referências

Angi, A., Sacco, A., and Marchetto, G. (2025). LLNeT: An intent-driven approach to instructing softwarized network devices using a small language model. *IEEE TNSM*.

Fuad, A., Ahmed, A. H., Riegler, M. A., and Čičić, T. (2024). An intent-based networks framework based on large language models. In *IEEE NetSoft*, pages 7–12.

Hollósi, G., Ficzere, D., and Varga, P. Generative AI for low-level NETCONF configuration in network management based on YANG models. In *IEEE CNSM*, pages 1–7.

Hong, J., Tu, N., and Hong, J. A comprehensive survey on LLM-based network management and operations. 35(6):e70029.

Jeong, E.-D., Kim, H.-G., Nam, S., Yoo, J.-H., and Hong, J. W.-K. S-witch: Switch configuration assistant with LLM and prompt engineering. In *IEEE NOMS*, pages 1–7.

Lin, J., Dzeparoska, K., Tizghadam, A., and Leon-Garcia, A. AppleSeed: Intent-based multi-domain infrastructure management via few-shot learning. In *IEEE NetSoft*.

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

Liu, F., Farkiani, B., and Crowley, P. A survey on large language models for network operations & management: Applications, techniques, and opportunities.

Mekrache, A. and Ksentini, A. (2024). Llm-enabled intent-driven service configuration for next generation networks. In *IEEE NetSoft*, pages 253–257.

Mekrache, A., Ksentini, A., and Verikoukis, C. (2024). Intent-based management of next-generation networks: An llm-centric approach. *Ieee Network*, 38(5):29–36.

Mondal, R., Tang, A., Beckett, R., Millstein, T., and Varghese, G. What do LLMs need to synthesize correct router configurations? In *22nd ACM HotNets*, pages 189–195.

Tageldien, M., Selim, B., and Sboui, L. (2025). Large language models in intent-based networking: a comprehensive survey across the intent lifecycle. In *ITC-Egypt*. IEEE.

Tu, N., Nam, S., and Hong, J. W.-K. (2025). Intent-based network configuration using large language models. *International Journal of Network Management*, 35(1):e2313.

Wang, C., Scazzariello, M., Farshin, A., Kostic, D., and Chiesa, M. Making network configuration human friendly.

Wei, Y., Xie, X., Hu, T., Zuo, Y., Chen, X., Chi, K., and Cui, Y. (2025a). Inta: Intent-based translation for network configuration with llm agents. In *IEEE 33rd ICNP*, pages 1–16.

Wei, Y., Xie, X., Zuo, Y., Hu, T., Chen, X., Chi, K., and Cui, Y. (2025b). Leveraging llm agents for translating network configurations. *arXiv preprint arXiv:2501.08760*.

[1]`https://www.rnp.br`
[2]`https://gt-iotedu.github.io`
[3]`https://www.gov.br/capes/pt-br`