Poster: Multi-Agent LLM System for Cisco Router Configuration

Michal Rozsíval
Brno University of Technology
Brno, Czech Republic
irozsival@fit.vut.cz

Petr Matoušek
Brno University of Technology
Brno, Czech Republic
matousp@fit.vutbr.cz

Jaromír Kotala
Brno University of Technology
Brno, Czech Republic
xkotal01@stud.fit.vut.cz

***Abstract*—Every network device has a configuration file containing the current settings and operational functionality such as IP addresses, running routing processes, and filtering rules. When new functionality is requested, the network administrator updates the configuration file by adding new commands. This process can be automated using an LLM model that understands the configuration language and can generate the relevant configuration commands. This paper presents a multi-agent LLM system that generates network configurations. Our approach uses a sequence of LLM-based agents to decompose the original configuration task, expressed in natural language, into logical subtasks, which are then translated into configuration commands. The proposed LLM-based configuration generation process includes validation and suggests sanity tests to eliminate potential errors. We demonstrate our approach by generating Cisco IOS configuration files for multiple devices. We also propose a dataset of non-trivial reference configurations for evaluating generation accuracy.**

***Index Terms*—Network configuration, LLM models, Cisco IOS, LLM agents, validation.**

I. Introduction

When setting up a new network device or adding functionality to an existing network, an administrator typically carries out basic configuration steps, which include setting a hostname, assigning an IP address, configuring basic device security, starting the routing process or implementing filtering.

Replacing the role of administrator with an LLM model that can generate the requested configuration based on our intentions seems to be straightforward. However, LLM models are probabilistic tools, meaning the generated output is not precise. Instead, the best possible solution is provided based on LLM parameters and training datasets. While this approach works well for text generation, it can cause serious problems with configuration files, where even a minor typo or the incorrect placement of a command can lead to a significant error. Therefore, we must develop a strategy to instruct the LLM model to generate the correct configuration and verify the syntax and semantics with respect to the required task. The fact that the configuration tasks can be implemented in various ways presents a new challenge: how to evaluate the generated results when two different implementations are possible?

This study addresses the aforementioned challenges by developing a multi-agent LLM system that decomposes the given task into a sequence of steps and atomic instructions. We guide the LLM model to generate the relevant configuration by providing task-related hints. To verify that the generated configuration meets the task requirements, we have implemented a catalogue of validation functions. The validation agent selects the relevant functions and verifies the generated configuration.

Supported by the project "Novel AI-Driven Process Automation for Simplifying and Enhancing Telecommunication Processes (Spojit)", 2024-2026, no. FW10010014, funded by the Technological Agency of the Czech Rep.

A. Contribution

The main contribution of this paper is the design and implementation of a prototype multi-agent LLM configuration system. Based on our experiments, we provide recommendations on pre-processing tasks, instructing the LLM to generate a proper configurations and verifying the output. We also provide a dataset of configuration tasks with solutions that can be used to evaluate the accuracy of network configuration generation.

II. Related Work

Several research studies have discussed the application of LLM in telecommunications [1], intent-based networking [2], [3], network configuration analysis [4] and translation [5], [6].

Mondal et al. [7] examine the ability of LLM models to generate and translate a simple configuration from Cisco to Juniper. They combine the GPT-4 model with verifiers to provide localised feedback and automatically correct errors. The configuration generation process runs in loops until the generated output is correct based on manual inspection. For verification purposes, they use a Batfish syntax verifier [8] alongside with Campion [9] and a topology verifier.

Another study of Chakraborty et al. [10] applies an LLM to generate a network configuration for an ISP network based on a tariff plan. The LLM analyses the tariff plan written by a human and identifies keywords with their values. These keyword-value pairs are then mapped to parameters in a predefined API template. The orchestrator then determines, which network functions to apply and invokes the corresponding function APIs. Finally, a human operator then checks the configuration.

Wei et al. [11] focus on translating the network configurations using an LLM agent. They employ intent-based RAG (iRAG), which systematically splits configuration files into fragments, extracts intents, and generates accurate translations. The generated configuration is analysed using syntax and semantic verification. Syntax verification uses a command syntax tree and a configuration parser, whereas semantic verification analyses configuration differences using an LLM.

III. Architecture of the Multi-Agent LLM System

The architecture of the proposed multi-agent LLM system emulates the decision-making process of a human network expert who would provide configuration updates. This involves analysing the specified tasks, decomposing them into single configuration steps, updating relevant configuration files and verifying the implemented changes.

A. Preprocessing Phase

The LLM agent breaks a user-specified task down into a series of self-contained subtasks, each representing a specific modification intention. This streamlined approach ensures that no aspect of the task is overlooked during subsequent processing. The agent then expands each subtask into a detailed sequence of high-level steps outlining the general workflow that the operator must follow to complete it. The aim is to explicitly detail all the necessary steps, eliminating the need for further processing to account for any implicit assumptions.

B. Generation Phase

The LLM agent filters the relevant configurations and selects those containing essential contextual information or crucial restrictions for fulfilling the subtask. It then determines an update order that preserves consistency and develops a plan for updating the current configuration, using high-level instructions derived from the detailed steps relating to the current subtask and the context. Finally, the agent applies the previously generated high-level instructions to the current configuration. This process may involve translating the instructions into specific operating system commands or removing irrelevant commands from the configuration file. This process is repeated for each subtask until they are all processed.

C. Validation Phase

Each configuration is validated after an update to ensure it is consistent with the instructions outlined in the task description. Once all configurations have been updated, their overall integrity in relation to the associated subtask is validated. This validation process uses a database of manually created validation functions and employs the pyBatfish network validation tool [8]. Firstly, the LLM agent determines the relevance of each validation function. It then generates the call signature for each selected validation function based on its description. Furthermore, the LLM agent uses a dedicated coverage analysis process to assess whether the chosen validation functions comprehensively address all aspects of the updates comprehensively. It generates a report that highlights any elements that may have been overlooked. This complementary approach ensures that each modification is either validated or reported to the user as unverified, along with a suggested validation strategy.

D. Testing Phase

The LLM agent generates sanity tests relating to the initial task. This allows users to systematically verify that the generated configurations conform to the original requirements.

IV. Dataset and Experiments

A prototype of a multi-agent LLM system was created as Python scripts communicating with LLM APIs (ChatGPT 5.0, Llama). The code along with the testing scenarios, and results, is available at github.com/MichalRozsival/configuration-agent.

V. Discussion and Future Work

The work and experiments presented show that it is feasible to replace a significant portion of a network admin's mundane work with LLM models. Despite the common limitations of LLM models, such as instability, hallucinations and a lack of domain-specific training datasets, we can generate valid network configurations based on requests in natural language.

The following observations help to achieve good accuracy:

- Transform a task into a specific instructions (sub-tasks).
- Decompose the sub-tasks into single configuration steps.
- Guide the LLM to generate the configuration for a single step, providing the full context of the task.
- Validate the generated configuration using tools such as pyBatfish, or create your own validation functions.
- Provide an explanation for each step of the process.
- Provide sanity test commands that can prove the deployed configuration meets the given task.

All of the above mentioned steps can be implemented using LLM models, for which we provide task-specific guidance.

Future work will include a support for multiple file configurations, enhancements through domain-specific Retrieval Augmented Generation (RAG) and configuration templates, and and extension of typical configuration scenarios.

References

[1] H. Zhou, C. Hu, Y. Yuan, Y. Cui, Y. Jin, C. Chen, H. Wu, D. Yuan, L. Jiang, D. Wu, X. Liu, C. Zhang, X. Wang, and J. Liu, "LLM for Telecommunications: A Comprehensive Survey on Principles, Key Techniques, and Opportunities," 2024.

[2] A. Mekrache and A. Ksentini, "Llm-enabled intent-driven service configuration for next generation networks," in *2024 IEEE 10th Int. Conf. on Network Softwarization (NetSoft)*. IEEE, 2024, pp. 253–257.

[3] K. Dzeparoska, J. Lin, A. Tizghadam, and A. Leon-Garcia, "LLM-Based Policy Generation for Intent-Based Management of Applications," in *19th Int. Conf. on Network and Service Management*, 2023, pp. 1–7.

[4] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proc. of the Conf. of the Special Interest Group on Data Communication*. ACM, 2017, p. 155–168.

[5] Y. Wei, X. Xie, Y. Zuo, T. Hu, X. Chen, K. Chi, and Y. Cui, "Leveraging llm agents for translating network configurations," *arXiv preprint arXiv:2501.08760*, 2025.

[6] N. Zheng, F. Li, Z. Li, Y. Yang, Y. Hao, C. Liu, and X. Wang, "Configtrans: Network Configuration Translation Based on Large Language Models and Constraint Solving," in *IEEE ICNP Conf.*, 2024, pp. 1–12.

[7] R. Mondal, A. Tang, R. Beckett, T. Millstein, and G. Varghese, "What do LLMs need to Synthesize Correct Router Configurations?" in *Proc. of the 22nd ACM Workshop HotNets*, New York, USA, 2023, p. 189–195.

[8] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *12th USENIX Conference on NSDI*, 2015, p. 469–483.

[9] A. Tang, S. K. R. Kakarla, R. Beckett, E. Zhai, M. Brown, T. Millstein, Y. Tamir, and G. Varghese, "Campion: debugging router configuration differences," in *ACM SIGCOMM 2021 Conference*, 2021, p. 748–761.

[10] S. Chakraborty, N. Chitta, and R. Sundaresan, "Automation of Network Configuration Generation using Large Language Models," in *20th Int. Conf. on Network and Service Management*, 2024, pp. 1–7.

[11] Y. Wei, X. Xie, Y. Zuo, T. Hu, X. Chen, K. Chi, and Y. Cui, "Leveraging llm agents for translating network configurations," 2025.