Flow-Rule Generation for SDN Using LLMs with Retry-Based Deployment Validation

Anouar El Hachimi[1*], Nicola Di Cicco[1], Memedhe Ibrahimi[1], Francesco Musumeci[1], and Massimo Tornatore[1]
[1]*Politecnico di Milano, Milan, Italy*; [*]Corresponding author: *anouar.elhachimi@polimi.it*

***Abstract*—This work proposes a pipeline for Software Defined Networking (SDN) that enables natural-language–based flow rule configuration using large language models (LLMs). The system addresses two key challenges: 1) the ambiguity and incompleteness of natural language inputs, and 2) the difficulty of reliably translating them into deployable SDN configurations. To this end, the pipeline integrates: *i)* an intent recognition module that refines user prompts via iterative clarification, and *ii)* a retry-based correction mechanism that handles failed configurations by regenerating and resubmitting corrected versions. These components are combined with intermediate YAML generation, documentation-based enrichment, and final translation into OpenFlow-compliant JSON for Ryu controllers. The pipeline is evaluated on flow rule deployment tasks of varying complexity, achieving an accuracy up to 96.7%, while maintaining cost-efficiency with an estimated API cost of only $0.08 per 100 configurations and remaining model model-agnostic.**

***Index Terms*—Software Defined Networking (SDN), Flow Rule Automation, Ryu Controller, OpenFlow 1.3, Large Language Models (LLMs), YAML-to-JSON Translation, Retrieval-Augmented Generation (RAG), Intent-Based Networking, REST API, Deterministic Retry Mechanism.**

I. Introduction

Software Defined Networking (SDN) has significantly transformed network management and control by enabling programmability, centralization, and dynamic policy enforcement [1]. Despite these advancements, configuring SDN systems, such as defining flow rules in OpenFlow-based environments, still requires considerable technical expertise [2]. Crafting syntactically and semantically correct flow entries demands a detailed understanding of protocol fields, device-specific configurations, and controller interfaces. This complexity not only restricts SDN programmability to expert users but also slows down operational agility.

In parallel, the emergence of large language models (LLMs) has opened new avenues for network automation by enabling the translation of high-level user intents into low-level configurations. However, applying LLMs to SDN tasks presents a critical challenge: these models frequently generate unreliable or hallucinated outputs, which poses a substantial risk when configuring production networks. While recent studies have begun exploring LLMs in networking [3], [4] [5], most lack built-in mechanisms for error detection or correction, limiting their practicality for real-world deployment.

To overcome these limitations, this paper proposes a reliability-focused, LLM-driven pipeline for flow rule deployment in Ryu SDN environments. The proposed approach combines three key components: (1) prompt clarification, (2) retrieval-augmented generation (RAG), and (3) a retry-and-correction mechanism to enhance the robustness of generated configurations. Given a user intent (e.g., "drop all IPv4 packets"), the pipeline not only produces valid flow rules but also verifies their correctness post-deployment and autonomously rectifies errors through iterative retries.

Consider, for example, a network operator issuing the intent: "drop all IPv4 packets." While semantically clear, this instruction omits critical fields such as the flow rule's priority. Our pipeline detects such ambiguities and responds by initiating a clarification loop, where the LLM explicitly requests missing parameters like the `priority`. After gathering the necessary information, the model generates a structured YAML representation of the flow rule and presents it to the user for review. This intermediate step allows the user to verify the configuration, and if needed, request corrections—such as adjusting the priority or modifying match conditions—before the pipeline proceeds to the final deployment phase.

By combining natural-language processing with domain-specific validation, the proposed pipeline enables SDN configuration from high-level user intent. Experimental results across flow rule tasks of increasing complexity confirm the system's effectiveness, achieving up to **96.7%** deployment accuracy, even in the presence of structurally complex inputs.

II. Background and Related Work

A. Software Defined Networking and Flow Rule Management

SDN decouples the control plane from the data plane, enabling centralized and programmable network management [1]. Among SDN protocols, OpenFlow is widely adopted as the southbound interface for communication between controllers and switches. Popular SDN controllers such as Ryu, ONOS, and OpenDaylight expose APIs that allow dynamic flow rule management [6].

Figure 1 illustrates the layered SDN architecture adopted in our system. In conventional setups, flow rules are manually written and submitted to the SDN controller via REST APIs. These rules are typically formatted in JSON and must conform to the controller's schema and the OpenFlow specification. Once received, the controller such as Ryu interprets the rule and installs it on the appropriate network devices through the southbound interface. In our system, this process is automated: user intents, expressed in natural language, are parsed and translated by an LLM-based pipeline into JSON configurations, which are then submitted to the controller for deployment.

978-3-903176-75-1 ©2025 IFIP

Although SDN enables fine-grained control over packet forwarding, the configuration of flow rules remains a technically complex task. Each rule must adhere to the strict syntax and semantics defined by the OpenFlow specification, including correctly formatted match fields, actions, and priority values [1], [2]. As network scale and heterogeneity increase, the number and diversity of flow rules grow accordingly, making manual configuration both error-prone and inefficient [7].

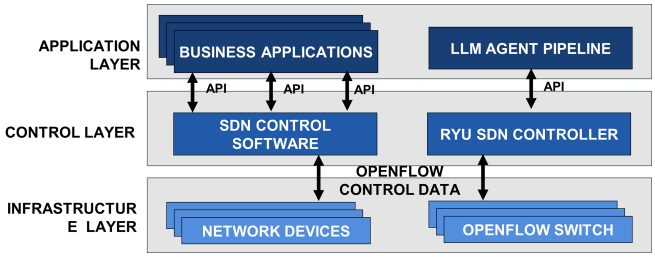

Fig. 1. High-level architecture of our LLM-assisted SDN pipeline integrating user intent, REST API-based rule translation, and OpenFlow rule installation.

B. Natural Language Interfaces and LLMs in Network Automation

The use of natural language interfaces in networking is gaining momentum, particularly as a means to abstract technical complexity for operators [8]. Initial research in this area has focused on intent-based networking (IBN), where high-level policy statements are compiled into low-level network configurations [9] [10].These systems, however, often rely on predefined grammars or domain-specific languages (DSLs), limiting their flexibility and extensibility. In contrast, our work is not a replacement but a complement: while IBN/DSL frameworks provide structured and reliable pipelines, we investigate whether LLMs can extend usability by enabling natural language as a more flexible interface, handling ambiguity while ensuring schema correctness.

Recent advances in LLMs, such as GPT-4 and Google Flash, have demonstrated remarkable capabilities in structured generation, code synthesis, and prompt-based control flow [11]. Frameworks like LangChain [12] allow for the composition of modular pipelines involving retrieval, reasoning, and structured output generation. In the context of communication networks, LLMs have been applied to tasks such as documentation, log analysis, and conversational troubleshooting [13]. However, their integration into closed-loop control pipelines with built-in validation checks, retry mechanisms, and automatic error correction for flow rule deployment (such as the one proposed in this paper) remains largely unexplored.

C. Novelty of the Proposed Work

To the best of our knowledge, this work is the first to introduce an LLM-agnostic pipeline for SDN flow rule deployment that combines 1) an LLM-based intent recognition module, capable of resolving ambiguous or partial natural language prompts through iterative clarification, and 2) a retry-based mechanism, namely, the *retry module*, for correcting failed configurations, regenerate and resubmit revised versions of the flow rule configuration. These two modules are the core of the pipeline, enabling handling of incomplete or incorrect user inputs. Unlike prior work [14] that focuses on static intent translation or predefined rule templates, our system dynamically enriches and disambiguates input to produce semantically complete YAML configurations.

III. LLM-Based Flow Rule Automation Pipeline

This section presents the architecture and internal logic of our modular pipeline, which enables the automatic generation and deployment of OpenFlow rules on the Ryu SDN controller from natural language prompts. Figure 2 provides a visual representation of the pipeline.

A. Pipeline Overview

The pipeline is composed of five primary stages spanning three architectural layers: Application, Control, and Infrastructure.

1) **User Input Interface:** A network administrator interacts with the system by submitting a natural language prompt describing the intended network behavior (e.g., “block all incoming IPv4 traffic”). This component captures both the raw prompt and a set of *contextual metadata*, which may include the datapath identifier (DPID) and the `table_id`. Notably, these contextual elements are treated as fixed inputs external to the user’s specific request and are injected upstream to improve the completeness and accuracy of downstream parsing.
2) **Intent Recognition Module:** The LLM interprets the user’s initial natural language prompt and, if key parameters (e.g., priority, IP fields) are missing, it proceeds by applying default values. It then generates a preliminary YAML representation of the flow rule, shown in Figure 2, and presents it to the user for review. The user can either accept the configuration or request specific changes, for example, modifying the ‘priority‘ or add new criteria. This clarification loop continues until the user explicitly approves the YAML. For instance, if a user inputs “create a flow rule to drop all IPv4 traffic,” the system may assign a default priority of ‘100‘ and generate a rule that matches on ‘ethType: 2048‘ with no actions (to drop packets). If the user replies “make priority 50,” the YAML is updated accordingly. Only after confirmation from the user, the pipeline proceed to the next stage.
3) **Retrieval-Augmented Generation:** The confirmed YAML file is then passed to a LangChain-powered Retrieval-Augmented Generation (RAG) module. This stage dynamically enriches the LLM prompt by retrieving technical documentation, specifically, structured text extracted from the official Ryu controller web documentation [15]. This knowledge base provides critical information about acceptable field formats, required JSON schema, and common usage patterns. The augmented

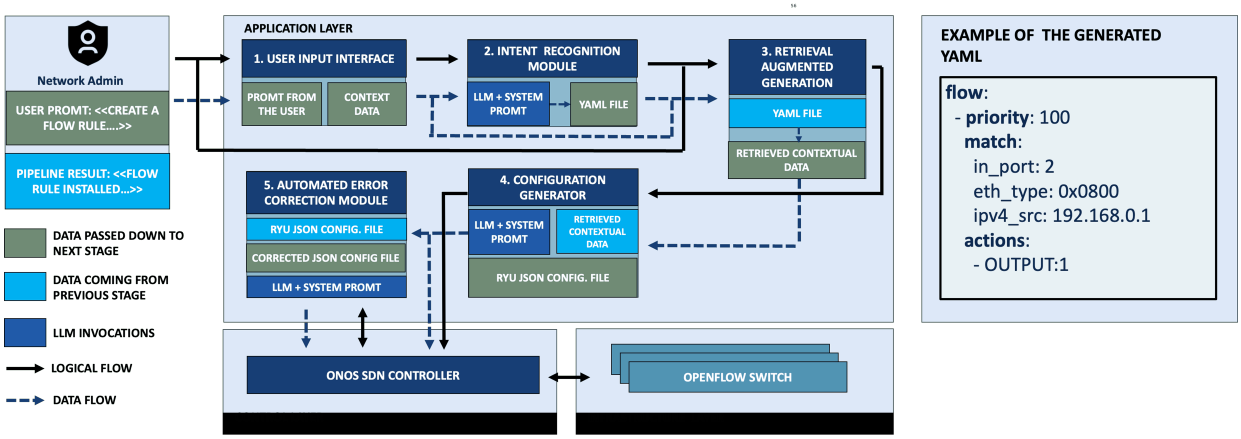

Fig. 2. End-to-end architecture of the proposed LLM-based SDN pipeline. The user issues a prompt which is incrementally transformed, validated, and installed as a valid OpenFlow rule in Ryu. On the right, an LLM-generated YAML file shows the intermediate structured configuration inside the pipeline.

prompt enables the model to generate fully conformant, context-aware JSON configurations for the Ryu API.

4) **Configuration Generator:** The enriched YAML and retrieved documentation are then processed by a second LLM. The model is prompted to synthesize a valid JSON object conforming to Ryu's API schema. The JSON includes match fields, actions, priority, device ID, and table ID.
5) **Retry Module:** After the JSON is generated, it is submitted to the Ryu controller via a HTTP POST request. If the response indicates failure (i.e., response code other than 200), the system enters an automated retry loop resending the request up to three times.This limit is based on empirical evidence showing that additional attempts beyond three yield negligible improvement in success rate. If the rule is still not successfully installed or the resulting flow table does not reflect the original user intent, a correction loop is triggered. This loop uses another LLM to revise the JSON by comparing the installed flow table with the expected YAML specification. The corrected version is again submitted, with up to three additional retries. If all retry and correction attempts fail, control is returned to the user, along with a detailed explanation (in natural language) and the last attempted configuration, minimizing the need for human manual intervention.

B. End-to-End Execution Logic

At a high level, the system executes the flow rule installation process as follows:

1) The user submits a natural language prompt specifying the intended flow rule behavior. Contextual metadata is appended to the request; this includes the selected datapath identifier (DPID) and the target `table_id`.
2) The prompt is parsed by an LLM-powered intent recognizer. If the input is incomplete or ambiguous, the system initiates an interactive dialogue to collect the missing information.
3) Once validated, a YAML representation of the flow rule is generated and presented to the user for inspection and optional manual refinement.
4) The finalized YAML file is passed to the RAG module, which enriches the system prompt with domain-specific knowledge retrieved from the official Ryu REST API module documentation [15]. This guarantees consistency with OpenFlow constraints and schema compliance.
5) The language model generates a JSON configuration suitable for the Ryu SDN controller's REST API.
6) The JSON object is submitted via HTTP POST to the controller endpoint. The system waits for the response and records the outcome.
7) **Retry Module:** If the controller returns a non-success status code (non-2xx), the retry module is automatically activated—without requiring human intervention. This module autonomously performs: 1) Extraction and analysis of error feedback from the controller response 2) Prompt regeneration and correction of fields likely causing the failure 3) Up to three controlled retry attempts, spaced with a fixed delay to account for controller latency

IV. Numerical Results and Evaluation Settings

A. Emulation Environment

To test the pipeline in a realistic and controllable SDN environment, we used Mininet [16] to emulate the network topology and OpenFlow switches. Mininet provides a lightweight virtual network that supports standard SDN protocols and integrates seamlessly with the Ryu controller. Each test scenario was instantiated as a custom Mininet topology with one or more switches, allowing for consistent deployment of flow rules and accurate monitoring of controller-switch interactions during evaluation.

B. Accuracy for Varying Flow-Rule Complexity

1) Flow Rule Complexity: As a first step in evaluating the robustness of our pipeline, we introduce a scalar **complexity**

score C to characterize the difficulty of each YAML-defined flow rule. This score is calculated as the total number of conditions used to match packets (M) and the number of actions applied when a match occurs (A), such that $C = M + A$.

Based on this score, we classify flow rules into three complexity tiers: **Simple** ($C \leq 3$), **Moderate** ($4 \leq C \leq 6$), and **Hard** ($C \geq 7$).

For example, a rule that matches on two packet properties (such as protocol type and destination port) and applies one action (such as forwarding the packet) has a complexity score of $C = 2 + 1 = 3$, and is thus classified as **Simple**.

This metric enables consistent benchmarking of how structural and operational complexity affects LLM performance. Higher C values imply greater disambiguation needs, schema issues, and retries, making it a practical baseline.

2) Accuracy: To complement the complexity score and provide a measure of practical effectiveness, we define a second metric: **accuracy**. This metric focuses on the success rate of deploying flow rules under real conditions. We use this metric to assess the system's performance across the three complexity levels and with different LLM backends.Accuracy is defined as the percentage of flow rules that were successfully installed on the SDN controller within the allowed retry attempts, i.e., at most three, while conforming to the user's original intent. Formally, we define the task accuracy A as:

$$A = \frac{N_{\text{success}}}{N_{\text{total}}} \times 100 \tag{1}$$

where N_{success} is the number of correctly installed flow rules (confirmed via flow table verification), and N_{total} is the total number of flow rules submitted to the system.

Figure 3 reports the overall success rate achieved by two leading LLMs (GPT-4o-mini and Google Gemini Flash 1.5) across the three predefined rule complexity levels. Accuracy values are averaged over 35 unique test cases per complexity tier, totaling 105 tasks per model. We generate each input by randomly sampling a syntactically valid YAML flow rule, categorized by our complexity metric. This choice allows us to bypass ambiguity in natural language parsing and directly evaluate the system's ability to validate, translate, and deploy rules end-to-end. As expected, accuracy decreases with increasing flow rule complexity. GPT-4o-mini outperforms Gemini Flash across all complexity levels, with a top Success rate of 96.7% for simple tasks and a worst-case accuracy of 73.3% on high-complexity rules. Error bars indicate 95% confidence intervals over the sampled tasks.

C. Completion@k Analysis

To evaluate the impact of the retry module, we compute the metric *Completion@k*, which measures the cumulative percentage of tasks completed in k or less retries, for $k \in \{1, 2, 3\}$. We define *Completion@k* as:

$$C@k = \frac{N_{\text{completed@}k}}{N_{\text{total}}} \times 100 \tag{2}$$

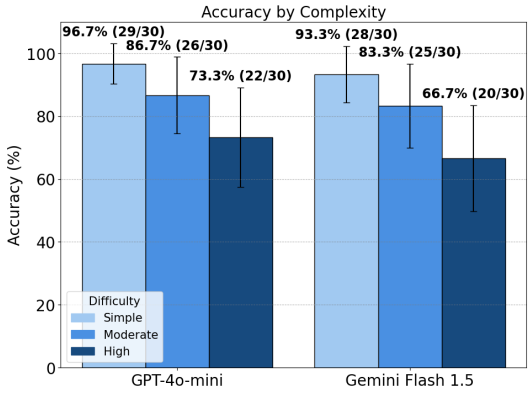

Fig. 3. Accuracy by complexity and LLM model (with 95% C.I.).

where $N_{\text{completed@}k}$ is the number of tasks successfully completed within k or fewer retry attempts, and N_{total} is the total number of tasks evaluated.

TABLE I
CUMULATIVE COMPLETION@K RESULTS FOR GPT-4O-MINI

Difficulty Level	*Completion@1*	*Completion@2*	*Completion@3*
Simple	93.3% (28/30)	96.7% (29/30)	96.7% (29/30)
Moderate	76.7% (23/30)	86.7% (26/30)	86.7% (26/30)
High	50.0% (15/30)	73.3% (22/30)	83.3% (25/30)

Each value shows the percentage and count of completed tasks within k retries.

We observe that, while most low-complexity rules are successfully completed on the first attempt, high-complexity rules substantially benefit from retries. For instance, *Completion@3* for high complexity reaches 83.3% (25/30), with only 50.0% of the rules being completed without retries due to increased schema requirements and a higher chance of field-level inconsistencies.

D. Resource Utilization and Cost Analysis

To evaluate the operational scalability of the proposed LLM-assisted pipeline, we analyze the resource consumption associated with each task, focusing on LLM token usage, where a token is a basic unit of text used by language models, typically representing a word or part of a word, and the resulting projected API costs. For each natural language prompt, the system performs:

- An initial intent parsing and clarification dialogue
- A RAG step using technical documentation
- Final JSON generation and potential retry/correction calls

We recorded the **average input tokens per rule** (sum of tokens sent across all calls), the **average output tokens per rule** (tokens produced by the model), and the **projected cost per 100 rules**, calculated by multiplying the average token counts by 100 and applying the pricing tiers.

Table II summarizes the average token consumption and cost per 100 flow rule tasks across different complexity levels.

This analysis demonstrates that the pipeline remains cost-effective for moderate-scale deployments (e.g., hundreds of

TABLE II
TOKEN USAGE AND ESTIMATED API COST PER 100 FLOW RULE TASKS (GPT-4O MINI)

Rule Complexity	Avg Input	Avg Output	Cost (USD/100)
Simple	4950 Tokens	98 Tokens	$0.0801
Moderate	5052.4 Tokens	217.2 Tokens	$0.0888
Hard	11276 Tokens	724.4 Tokens	$0.2126

Based on GPT-4o Mini pricing: $0.15/M input tokens, $0.60/M output tokens (June 2025) [17].

rules/day); however, infrastructure overhead is not considered, as it depends on factors such as energy prices, data center resource efficiency, and site-specific operational constraints.

E. Ablation Study

To assess the contribution of each pipeline component, we performed an ablation study by selectively removing the **Retry Module**, the **Retrieval-Augmented Generation (RAG)** stage, and the **Intent Recognition Module**, respectively. Figure 4 shows the resulting degradation in performance, broken down by rule complexity.

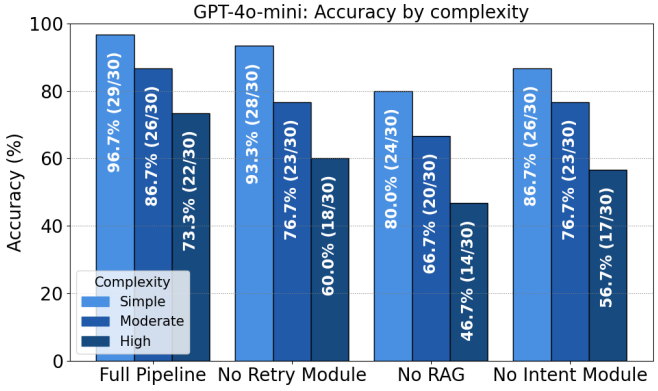

Fig. 4. Ablation results on GPT-4o-mini: performance by difficulty.

The full pipeline serves as a baseline. Notably, removing the Retry Module reduces Accuracy on high-complexity tasks from 73.3% to 60.0%. Eliminating the RAG stage causes an even sharper drop to 46.7%, indicating its essential role in ensuring schema correctness and semantic validity. The removal of the Intent Recognition Module shows moderate but non-negligible effects, especially in moderate and high tiers, where ambiguity and under-specification are more common.

V. CONCLUSION AND FUTURE WORK

This paper presents a modular, LLM-driven pipeline for natural language–based configuration of SDN flow rules. By integrating intent recognition, retrieval-augmented generation, and structured error recovery, the system enables robust, user-friendly interaction with Ryu-based OpenFlow controllers. The pipeline achieves high task accuracy (up to 96.7%) and demonstrates resilience under increasing rule complexity. The evaluation highlights the contribution of each component: RAG ensures schema conformity, the retry module enhances reliability in complex scenarios, and intent parsing guarantees semantic completeness from the outset. Together, these modules constitute a principled architecture for closing the gap between high-level intent and low-level SDN configuration. The pipeline is explicitly LLM-agnostic, allowing integration with both commercial and open-source models. For reproducibility, all API calls were executed using default parameter settings, ensuring consistent evaluation conditions. Future research will extend the pipeline to multi-controller and inter-domain environments, incorporate real-time feedback loops with live SDN telemetry (e.g., packet counters, congestion indicators), and investigate adaptive retry strategies beyond the fixed three-attempt policy to optimize robustness. In parallel, a broader security perspective will be developed, analyzing potential threats such as prompt injection and adversarial misuse of natural language inputs.

REFERENCES

[1] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (SDN): a survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833, 2016.
[2] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.
[3] N. Di Cicco, M. Ibrahimi, S. Troia, F. Musumeci, and M. Tornatore, "Open implementation of a large language model pipeline for automated configuration of software-defined optical networks," in *ECOC 2024; 50th European Conference on Optical Communication*, 2024, pp. 1591–1594.
[4] Wang et al., "Netconfeval: Can llms facilitate network configuration?" *Proc. ACM Netw.*, vol. 2, no. CoNEXT2, Jun. 2024. [Online]. Available: https://doi.org/10.1145/3656296
[5] A. Mekrache, A. Ksentini, and C. Verikoukis, "Intent-based management of next-generation networks: an llm-centric approach," *IEEE Network*, vol. 38, no. 5, pp. 29–36, 2024.
[6] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*. Lemesos, Cyprus: IEEE, Apr. 2016, pp. 1–6.
[7] S. Bhardwaj and S. N. Panda, "Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment," *Wireless Personal Communications*, vol. 122, no. 1, pp. 701–723, Jan. 2022.
[8] Huang et al., "Large language models for networking: Applications, enabling techniques, and challenges," *IEEE Network*, vol. 39, no. 1, pp. 235–242, 2025.
[9] A. Leivadeas and M. Falkner, "A survey on intent-based networking," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 625–655, 2023.
[10] L. Dinh, S. Cherrared, X. Huang, and F. Guillemin, "Towards end-to-end network intent management with large language models," 2025. [Online]. Available: https://arxiv.org/abs/2504.13589
[11] H. Naveed et al., "A comprehensive overview of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2307.06435
[12] H. Chase, "LangChain," Oct. 2022. [Online]. Available: https://github.com/langchain-ai/langchain
[13] M. Siino, F. Giuliano, and I. Tinnirello, "Llm application for knowledge extraction from networking log files," in *2024 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 2024, pp. 01–06.
[14] C. Wang, M. Scazzariello, A. Farshin, D. Kostic, and M. Chiesa, "Making network configuration human friendly," 2023. [Online]. Available: https://arxiv.org/abs/2309.06342
[15] Ryu SDN Project. (2025) ryu.app.ofctl_rest – rest api module. Open Networking Foundation. [Online]. Available: https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html
[16] Open Networking Foundation, "Mininet," http://mininet.org, 2019.
[17] OpenAI. (2025) Openai api pricing. [Online]. Available: https://openai.com/api/pricing