Open Implementation of a Large Language Model Pipeline for Automated Configuration of Software-Defined Optical Networks

Nicola Di Cicco*, Memedhe Ibrahimi, Sebastian Troia, Francesco Musumeci, Massimo Tornatore

Politecnico di Milano, Milan, Italy, *corresponding author: nicola.dicicco@polimi.it

Abstract *We leverage LLMs to develop a natural-language interface to a software-defined optical network testbed. Results show over 80% accuracy in translating human intent to the appropriate network configurations. Our code is public.*

Introduction

Large Language Models (LLMs) have experienced groundbreaking progress in recent years, achieving state-of-the-art performance in many natural language and code generation tasks[1]. In particular, tremendous advances in open-source LLMs make it possible to execute LLMs *locally* on commodity hardware while rivaling the performance of state-of-the-art proprietary solutions such as GPT-3.5[2],[3]. As such, LLMs hold vast untapped potential for developing innovative LLM-powered applications, with software-defined optical networking standing as no exception.

In this paper, for the first time to the best of our knowledge, we build and make public an LLM-based application realizing a natural-language interface to a software-defined optical-network (SDON) testbed. The purpose of the application is to translate open-ended natural language queries from a user (e.g., "set up a lightpath", or "measure the OSNR of a service") into domain-specific data models to be sent through an SDON, and to provide feedback on the outcome of the user's request. While our application targets specifically operations on our testbed, the methodology and the practical guidelines distilled in this paper are generalizable to any software-defined testbed exposing a well-defined Northbound Interface (NBI). We make our code (entirely built on open-source software and open-source LLMs), data, and scripts to reproduce experiments publicly available to foster future research in this field.[1]

Building an LLM-based interface for automated network configuration demands a specialized knowledge of the inner workings of both LLMs and the optical-network testbed. Here, we identify two fundamental challenges and anticipate our proposed working solutions to solve them.

Challenge 1: the LLM has zero prior knowledge about controlling any similar testbed. This is a sensible assumption, since domain-specific documentation of optical-network testbeds is, in general, not publicly available on the Internet, and hence cannot be present in the text corpora used to train the LLM. We therefore face the challenge of how to efficiently inject new knowledge into pre-trained LLMs.

Challenge 2: LLM outputs are not controllable (i.e., "hallucinations" occur). Regrettably, LLMs are prone to producing non-factual outputs. Though "prompt engineering" techniques (i.e., designing task-specific inputs) can mitigate this phenomenon, we have no guarantees that LLMs will never hallucinate. When interacting with a testbed, hallucinations can translate into malformed data in the best case, and catastrophically destructive actions in the worst case. For this reason, we need to implement fail-safe mechanisms to avoid undesired outcomes.

Teaching new skills to pre-trained LLMs

A first strategy for incorporating new knowledge into an LLM is through **fine-tuning** via Supervised Learning on a target training dataset. This has two main limitations: 1) it requires costly GPUs (though this need can be partially circumvented with modern approximating algorithms[4],[5]), and 2), most importantly, it requires sizeable datasets high-quality prompt-answer exemplars[4],[5], which may be extremely time-consuming to collect, e.g., in the case of instructions for SDON configuration.

Hence, we decided to adopt a second strategy, called **in-context learning**, i.e., all the information relevant to solving the new task is provided in the LLM's prompt. The intuition is that LLMs, being trained on colossal text corpora, become capable of *learning new skills from analogy* at runtime. One can think of prompting as a way of "programming" LLMs. Moreover, even though in-context learning is limited by the LLM's *context window* (i.e., its short-term memory), recent breakthroughs achieved context windows as long as 700.000 words[6]. However, even though in-context learning is an exceptionally powerful and versatile technique, naive prompting (e.g., "do X; context: Y") will result in sub-optimal outputs, mainly due to hallucinations[7]. In the following, we outline our proposed solutions to create valid outputs and minimize hallucinations.

[1] https://github.com/nicoladicicco/llm-orchestrator

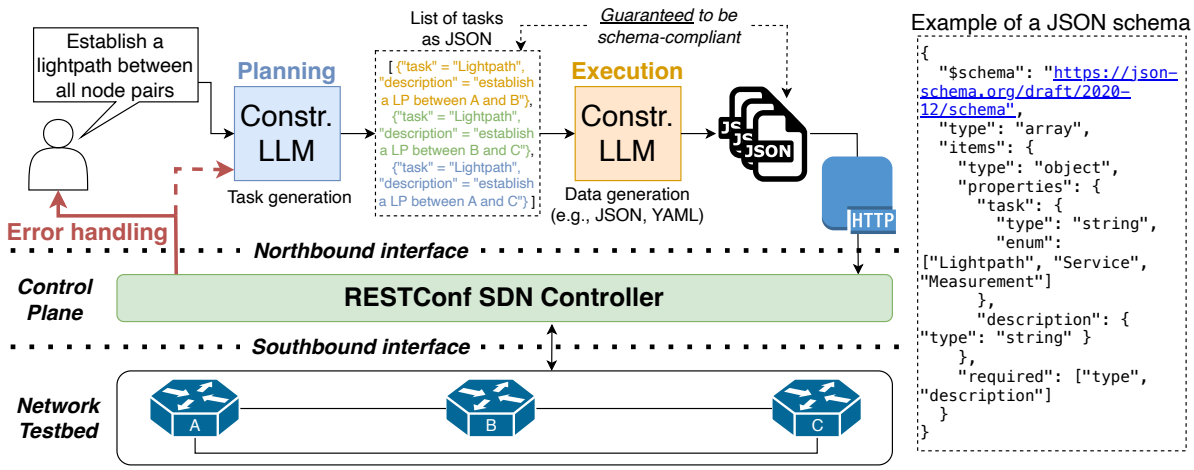

Fig. 1: System design of an LLM-based interface to a software-defined optical network testbed.

Designing an LLM-based interface for software-defined optical networks

Our core idea is as follows: while we cannot guarantee that the LLM's output will exactly realize the user's intent, we can guarantee that it will always be *valid*, i.e., well-formed, according to a user-specified data model. To achieve this goal, we leverage compositional learning[8]–[11], i.e., we decompose "translating human intent in an appropriate data structure" in three simpler tasks: 1) **Planning**, where an LLM translates human intent into a list of tasks, 2) **Execution**, where an LLM produces the appropriate data structure for each task, and 3) **Error handling**, where an LLM attempts to rectify easily-fixable errors with minimal human supervision. All phases leverage **Constrained Generation**, which ensures that the produced outputs are valid. Fig. 1 illustrates our proposed system design.

Step 1: Planning. We transform the user's query into a list of atomic tasks, which we represent as JSON data. We design a JSON schema with two fields: "task", which takes values in a finite set of keywords (in our case, "Lightpath", "Service" or "Measurement"), and "description", which is a summary in natural language of the task and of its requirements. Intuitively, we can think of the "prompt $\rightarrow$ task" mapping as sentence classification, and the "prompt $\rightarrow$ description" mapping as summarization/paraphrasing, both tasks at which LLMs naturally excel.

Step 2: Execution. We loop over each task generated in Planning and perform the following operations: (i) from the "task" field, we retrieve the corresponding data model (e.g., if "task" is "Lightpath", we retrieve a JSON schema defining the data model for instantiating a lightpath); (ii) we provide the "description" field alongside the retrieved data model to the LLM. The output is a data structure (e.g., a JSON list) realizing the task's requirements, which can then be encapsulated in a REST message and sent through the NBI.

Step 3: Error handling. Even though the messages generated by the LLM are syntactically correct, the SDN controller may not succeed in handling the generated requests (e.g., lightpath creation fails), returning an error. One option could be to return control to the user for manual intervention. Instead, we propose to let the LLM attempt to fix (some of) the errors. For instance, simple errors such as "lightpath ID 42 is already taken" can be easily fixed by choosing a different ID. As a simple solution, we can concatenate the error body to the input prompt, turning the error message into additional requirements, and restart generation from the planning phase. Note that most classes of errors (e.g., "no spectrum available between nodes A and B") cannot be trivially fixed without human guidance. In these cases, one can instruct the LLM to return control in case of specific error types, or (more challenging) let the LLM decide to return control if the input context is not sufficient to resolve the error. Our solution covers only a limited set of errors, leaving ample room for future research (e.g., in repeated user-LLM interaction.)

Constrained generation. We leverage *formal grammars* to constrain at runtime the generation process, such that each output is *guaranteed* to be a valid data structure under a user-specified data model. In particular, we leverage JSON Schemas, which we convert to GGML Bakus-Naur Form (GBNF) grammars[3]. During generation, we use the grammar to dynamically restrict the LLM's output dictionary according to the constraints specified by the JSON schema and the current generated text. As an example, consider the JSON schema in Fig. 1. When generating the field "task", we restrict the LLM's output to choose only among the specified admissible words, i.e., "Lightpath", "Service", or "Measurement". This grammar-based approach can be generalized to arbitrary domain-specific constraints (e.g., generating syntactically valid YAML files), providing an effective way for enforcing "guardrails" to the LLM's output.

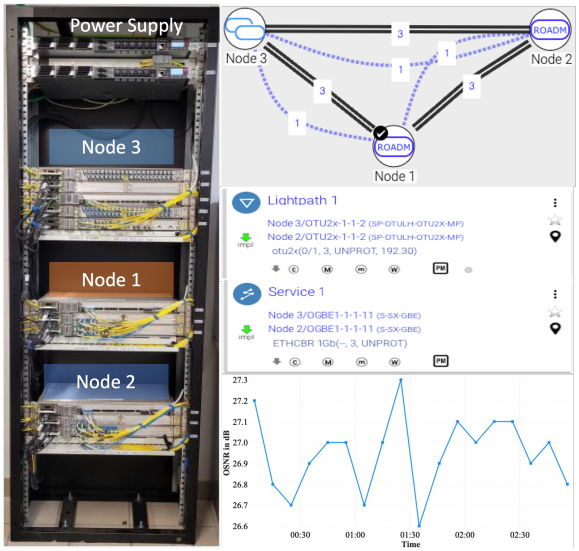

Fig. 2: Filterless optical network testbed. Solid and dashed lines represent fiber connections and lightpaths, respectively.

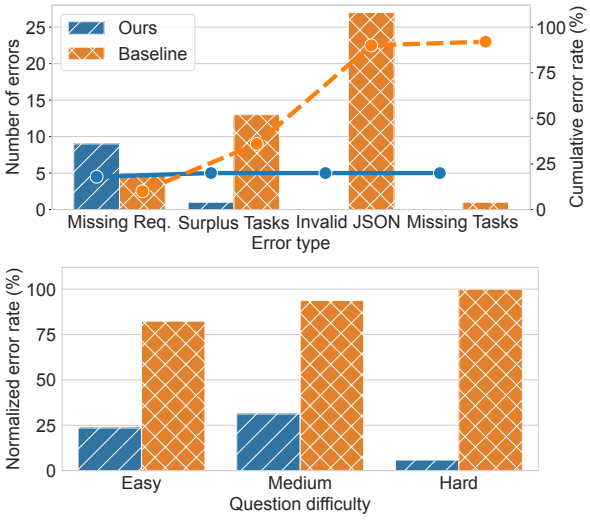

Fig. 3: Performance evaluation of our LLM-based pipeline for automated optical-network configuration.

Numerical Results

Testbed description. Our testbed is a three-node optical filterless ring network equipped with 10G (non-coherent) transponders. The testbed comprises WDM-layer equipment (to establish lightpaths between source-destination pairs) and OTN-layer equipment (to provision services that carry traffic at higher protocol layers), managed by the SDON controller and an orchestrator developed within our laboratory. Northbound and southbound interfaces are implemented using RESTConf. We consider three functionalities of the SDON controller, i.e., 1) lightpath establishment, 2) service provisioning, and 3) performance monitoring. Lightpaths can be established between any node pair through 10G transponders, while services carrying traffic may be 1G or 10G. Measurement campaigns can be initiated on established lightpaths and provisioned services, with 15-minute or 24-hour granularities. Fig. 2 illustrates our testbed. We show a successful implementation of *Lightpath 1* and *Service 1* between Node 2 and Node 3, and the OSNR measured at the receiver for *Lightpath 1*.

Performance evaluation. We use Mixtral-8x7B[2] as an LLM model. We manually curate a dataset including 50 prompt-answer pairs. The answers consist of JSON files realizing the question's requirements. Each question is marked as Easy, Medium, or Hard based on the length of the correct answer, as longer answers imply more tasks and/or requirements. We consider a generated answer to be "correct" if all the requirements are satisfied and "incorrect" if some requirements are not satisfied and/or the model hallucinates content beyond the user's request. Specifically, we consider four types of incorrect answers, i.e., "Invalid JSON", "Missing Tasks", "Surplus Tasks", and "Missing Requirements". We compare against a baseline leveraging in-context learning without task decomposition and constrained generation.

Fig. 3 shows our performance evaluation results. The baseline achieves a staggering 92% error rate, showing that LLMs are not plug-and-play tools for automated network configuration. In contrast, our approach achieves a 20% error rate. Moreover, while the baseline's performance gets worse with the answer's difficulty, our approach does not show a distinct trend. Indeed, the baseline has to generate long data structures attempting to respect the user's requirements and the JSON schema, while our approach generates a series of self-contained, small data structures. We then manually inspected the incorrect responses produced by our system. In eight out of nine "Missing Requirements" errors, the user's request could have been interpreted ambiguously. For instance, if the user asks "the ConfigurationState should be defined", it can either mean "the field ConfigurationState should be present in the JSON", or "set 'defined' as the value of the ConfigurationState field", with the latter being the intended interpretation. Minor paraphrasing solves these errors, reducing the total error rate to 4%. This leaves out two error instances: one where the model creates two services with the same name, and one where the model creates an unsolicited measurement campaign. The first error can be easily detected by the control plane and communicated with a message such as "Error: cannot create two services with the same name." By adding this message to the input prompt in the planning phase, the LLM generates a correct answer, achieving a 2% error rate. To conclude, another advantage of our task decomposition approach is inference time. On a high-end laptop, the baseline requires 90*s* to produce an output due to having to process a long input prompt. Instead, our approach requires less than 30*s* (a 3x speedup), with the main bottleneck being text generation.

Acknowledgements

This work was partly supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART") and by the PRIN project ZeTON, funded by Italian Ministry of University and Research.

References

[1] OpenAI *et al.*, *GPT-4 technical report*, 2024. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs.CL].

[2] A. Q. Jiang *et al.*, *Mixtral of experts*, 2024. DOI: 10.48550/arXiv.2401.04088. arXiv: 2401.04088 [cs.LG].

[3] G. Gerganov, *Llama.cpp*, https://github.com/ggerganov/llama.cpp/, 2024.

[4] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models", in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022.

[5] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms", in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 10 088–10 115.

[6] Google, *Gemini 1.5*, https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/, 2024.

[7] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, "Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts", in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023. DOI: 10.1145/3544548.3581388.

[8] J. Loula, M. Baroni, and B. Lake, "Rearranging the familiar: Testing compositional generalization in recurrent networks", in *EMNLP Workshop on BlackboxNLP*, 2018.

[9] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: Solving AI tasks with chatGPT and its friends in hugging face", in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[10] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models", in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2023.

[11] P. Lu, B. Peng, H. Cheng, *et al.*, "Chameleon: Plug-and-play compositional reasoning with large language models", in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 43 447–43 478.