Interface on demand: Towards AI native Control interfaces for 6G

Abhishek Dandekar
TU Berlin
Berlin, Germany
a.dandekar@tu-berlin.de

Prashiddha D. Thapa
Fraunhofer HHI
Berlin, Germany
prashiddha.dhoj.thapa@hhi.fraunhofer.de

Ashrafur Rahman
TU Berlin
Berlin, Germany
a.rahman@tu-berlin.de

Julius Schulz-Zander
Fraunhofer HHI
Berlin, Germany
julius.schulz-zander@hhi.fraunhofer.de

Abstract—**Traditional standardized network interfaces face significant limitations, including vendor-specific incompatibilities, rigid design assumptions, and lack of adaptability for new functionalities. We propose a multi-agent framework leveraging large language models (LLMs) to generate control interfaces on demand between network functions (NFs). This includes a matching agent, which aligns required control functionalities with NF capabilities, and a code-generation agent, which generates the necessary API server for interface realization. We validate our approach using simulated multi-vendor gNB and WLAN AP environments. The performance evaluations highlight the trade-offs between cost and latency across LLMs for interface generation tasks. Our work sets the foundation for AI-native dynamic control interface generation, paving the way for enhanced interoperability and adaptability in future mobile networks.**

Index Terms—**AI native, interfaces, 6G, agents, LLM**

I. Introduction

Mobile networks have evolved over time from being monolithic and tightly coupled to being more flexible. This started with the emergence of Software-Defined Networks (SDN) which separated the network control plane from the network data plane. This meant that instead of tightly coupled software and hardware, it was possible to run the software separately and remotely from the hardware. It paved the way for disaggregated networks. In disaggregated networks, instead of combining all functionality in a single Network Function (NF), it is split into multiple NFs. For example, in 5G networks, a single gNB can be split into Radio Unit (RU), Distributed Unit (DU), Control Unit - Control Plane (CU-CP), and Control Unit - User Plane (CU-UP) [1]. When a monolithic NF is split into multiple NFs, they require new interfaces to communicate between them. These interfaces can be control interfaces (e.g., F1-C) or user data interfaces (e.g., F1-U). The entire mobile network can be seen as a set of NFs that communicate and coordinate with each other over these interfaces.

In order to operate networks with disaggregated and distributed network functions, it is essential to define these network interfaces in advance. This is done by various standardization organizations (SDOs). However, using such predefined standardized interfaces also creates various issues.

- Vendors of all NFs may not be able to conform with these standardized interfaces, thus making multi-vendor deployment challenging.
- Standardized network interfaces are often specific to a particular Radio Access Technology (RAT) which means the NFs belonging to one RAT cannot communicate with another RAT.
- These interfaces are inflexible and designed assuming that the functionality of the NFs remains fixed. If a new functionality is to be added to NF, it cannot use the predefined network interface.

In order to address these challenges, we propose a framework where we can create a control interface between any two NFs on demand. We achieve this using a multi-agent framework based on Large Language Models (LLMs).

In Section II, we provide a brief background on LLMs, followed by an overview of our framework in Section III. Section IV explores one of the use cases realized using our framework, and Section V describes its implementation and evaluation. In Section VI, we discuss the current limitations of our framework. In section VII, we provide relevant research work and in the final section we conclude our work.

II. Background

A. Large Language Models

Large language models are machine learning models trained on vast quantities of textual data and are used to generate text in natural language [2]. These models are non-deterministic in nature and operate by predicting the next word in a sequence, iteratively constructing coherent and contextually relevant text. In order to generate an output these model needs to be provided with a set of instructions which are termed as *prompt*. A *context window* refers to the amount of text or data that the LLM can process and generate at a single time. LLMs have a finite context window hence the amount of text that could be generated and provided in a prompt is limited. In cases where it is necessary to provide a larger amount of text, techniques like Retrieval Augmented Generation (RAG) [3] are used.

Using RAG, the LLM utilizes retrieved information to generate outputs that are both factually grounded and also highly relevant and accurate. This approach significantly reduces the chances of producing incorrect or irrelevant responses while offering additional benefits such as enhanced adaptability across domains, real-time integration of the latest data, and improved user trust through source attribution. RAG ensures minimal hallucination and generates precise, reliable, and tailored outputs.

B. *Agentic LLMs*

An agent [4] can be seen as a set of software pipelines built on top of the LLM engine which allows access to external information, cross-checking, and reasoning. It is called an agent because it has *agency*, which is the ability act on its own without requiring constant input from a human. An agent can be specialized to perform specific tasks. In a multiagent system, multiple such specialized agents act together to achieve a certain goal. For example, an itinerary agent can create an itinerary for a trip based on user requirements. This agent can then forward the itinerary to a ticket booking agent which can fetch information from various airlines and book the most suitable ticket. Such multi-agent frameworks can be used to reach a certain goal from a set of requirements.

III. Interface Generation framework

This section provides an overview of the key steps required for generating a control interface between two NFs.

Consider two sets of NFs, a source NF (*NFsrc*) and a destination NF (*NFdest*) as shown in Figure 1. Let's assume *NFsrc* needs to generate a control interface towards *NFdest* in order to control its functionality. It uses two LLM based agents, Matching agent in *NFsrc* and Codegen agent in *NFdest* to generate this interface. Before the interface generation process can begin, *NFsrc* needs to discover *NFdest* using appropriate methods (e.g. DNS based discovery, static configuration, etc.). Subsequently, it needs to establish trust with *NFdest* using suitable mechanisms. Then a provisioning interface is setup between *NFsrc* and *NFdest* based on a predefined port. This interface is used by the respective agents to communicate with each other.

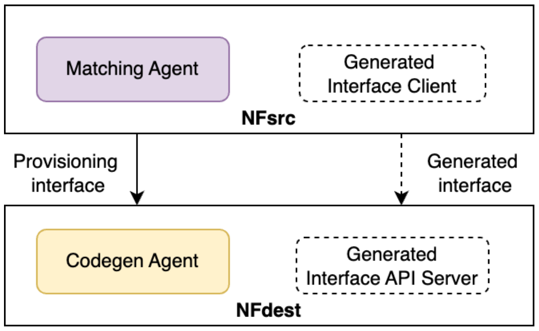

Fig. 1. Multi-agent framework

A. *Matching agent*

The initial step in generating interfaces is to meet two types of compatibility requirements. Firstly, both NFs need to be functionally compatible, which means that the target *NFdest* should have the capability to support the functionality required by the *NFsrc* (e.g. ability to change the transmission channel). Secondly, they need to be semantically compatible, this means that the *NFdest* needs to correctly understand the incoming control message from *NFsrc* irrespective of varying function names. For example, *NFdest* needs to understand that *set_channel* and *setchn* both refer to the same functionality of setting the channel number. Both NFs also need to have compatible encoding/decoding schemes for control messages (e.g. protocol buffer, flatbuffer, etc.).

When a user intends to send certain control functions from *NFsrc* to *NFdest*, it needs a mechanism to check if the required functionality is supported by the *NFdest*. To achieve this the user sends control function requirements to the matching agent (Fig. 2 step 1). The matching agent is an LLM based agent which allows the user to find whether the required control functions are supported and to identify the required input parameters. This is achieved by matching the required control function description from the user against the *NFdest* capability document (Fig. 2 step 2). The *NFdest* capability document details the capabilities of the *NFdest*. It can either be provided from an online repository of capability documents or directly from the *NFdest* itself.

If an exact match is not found among the NF control functions, the closest match is selected, which can later be *augmented* by the codegen agent. If the agent cannot find any similar function, it informs the user that the NF is unable to support required control functions. Note that this matching is done in terms of available functionality and not in terms of the exact API of the *NFdest*. This allows NF vendors to not expose their API externally.

If *NFdest* supports the required functionality then a control function requirements (CFR) document is created. This document contains a list of required control functions and their matching functions from capability document along with the required encoding scheme. It can include the required control function name, input parameters, output parameters, description of the control function and matched control function. It also includes details like parameter data types, units, etc. For example:

func setpower (radioID string, pow string dBm)(response boolean): <description >:<matched function>

The matching agent then generates an interface client which allows to send and receive control function messages to the *NFdest* (Fig. 2 step 3). The CFR document is then sent to the *NFdest* (Fig. 2 step 4). This document can be sent using a REST POST query over the provisioning interface.

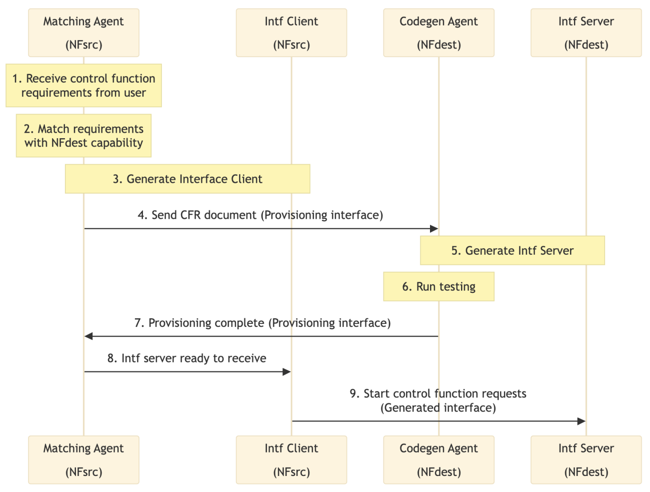
Matching Agent (NFsrc)
Intf Client (NFsrc)
Codegen Agent (NFdest)
Intf Server (NFdest)
1. Receive control function requirements from user
2. Match requirements with NFdest capability
3. Generate Interface Client
4. Send CFR document (Provisioning interface)
5. Generate Intf Server
6. Run testing
7. Provisioning complete (Provisioning interface)
8. Intf server ready to receive
9. Start control function requests (Generated interface)

Fig. 2. Interface generation workflow

B. *Codegen agent*

The CFR document sent from the matching agent is received by the codegen agent on *NFdest*. This agent is tasked with creating an interface API server app that can decode and execute the control functions provided in the CFR document. The codegen agent generates this interface API server using knowledge from an internal API document which is present on the *NFdest* (Fig. 2 step 5). The internal API document is a developer document specific to the vendor, detailing how to use the internal API to trigger the control functions supported by *NFdest*. Both the internal API and the API documentation can be proprietary and need not be exposed externally. The generated interface API server performs the following functions:

- Decode the incoming control message sent by the interface client from *NFsrc*
- If required adapt the unit or data type of control function parameters
- Trigger the corresponding internal APIs
- If required adapt the units and data types of parameters returned from the internal API
- Encode and send this information back to *NFsrc* interface client

In addition to this, the codegen agent can also generate augmented functions. This happens when the required control function does not have a direct matching function in the internal API. The codegen agent generates a wrapper function around the internal API function which is most similar to support the additional functionality required by the control function.

Once the code generation is done, the agent generates a test interface client code with dummy parameter values to check the validity of the generated interface server. This test code may also be provisioned by the *NFsrc*. If the interface server code throws errors in this test, this error is fed back to the LLM to regenerate this code. The agent can do it multiple times until the correct version of the code is generated (Fig. 2 step 6). Once the correct code is generated the codegen agent sends a provisioning complete message to the *NFsrc* to indicate that the new control interface is ready (Fig. 2 step 7). The matching agent then triggers interface client to initiate a connection (Fig. 2 step 8). After this step, the interface client initiates a connection with the interface server and starts sending control function requests (Fig. 2 step 9).

IV. Use case

In this section, we provide an illustrative example of how the multiagent framework described in previous sections can be used to generate interface on demand for controlling IEEE 802.11(WLAN) and 5G NFs on demand. O-RAN is a framework that disaggregated the 5G gNB-DU into O-DU, and O-RU. As per the principles of SDN, it also separates the control plane from 5G gNB and places it in two distinct controllers [5]. The Non-Real-Time RIC (Non-RT RIC) controls tasks with higher latency (>1s) while the Near-Real-Time RIC (Near-RT RIC) handles control tasks requiring lower latency (1s<). The Non-RT RIC sends control messages to the DU and the CU using the O1 control interface while the Near-RT RIC controls these NFs over the E2 control interface. Both these interfaces have been standardized by the O-RAN alliance. However, they exhibit the following issues:

- Managing a multi-vendor O-RAN deployment is challenging. Even though the DU/CU might use the standardized E2 interface, it may not be compatible with the near RT-RIC. This is due to divergence in implementation, and especially in the encoding schemes. A good example of this is the OSC-RIC [6] and FlexRIC [7]. As they have different E2 implementations, each DU/CU needs to have an OSC-RIC E2 agent and a FlexRIC E2 agent in order to communicate with the near-RT RIC. In a nutshell, the controllers and DU/CU are functionally compatible but they are semantically incompatible.
- Making changes to predefined, standardized interfaces like E2 requires changing the service module (SM) and also the low-level code of the DU/CU. This limits innovation as it slows the development cycles.
- 3GPP allows for using non-3GPP technologies like WLAN with 5G. However, SDOs like O-RAN do not specify a control interface towards WLAN, because of which joint control of 3GPP 5G and WLAN is not possible. These networks currently do not adapt to each other to achieve optimal resource utilization. In order to have joint control, we need a Multi-RAT RIC which can control both gNB and AP. With multi-RAT RIC, the RIC applications can coordinate and control both 5G and WLAN networks [8]. For instance, if the WLAN link provides more reliable latency and the 5G link offers higher throughput, resource allocation can be adjusted to prioritize latency on WLAN link and throughput on 5G link.

Figure 3 shows a scenario for joint deployment of 5G and WLAN. Note that there can be multiple APs and gNBs from various vendors. WLAN communicates with the 5G core network through N3IWF (N3 Inter working function) as defined by 3GPP [1]. The core network treats the WLAN Access Point (AP) as transparent and hence exerts no control over it. In order to control WLAN AP, multi-RAT RIC generates G2 and it generates G1 for controlling gNB.

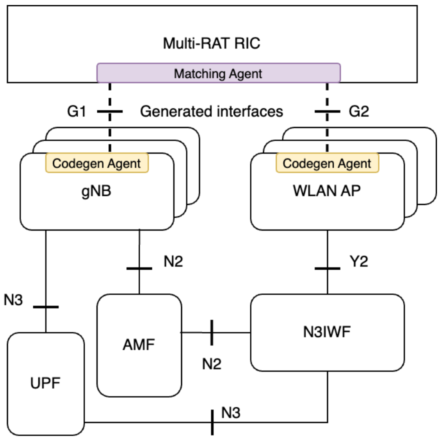

Fig. 3. Joint control for WLAN and 5G

Initially, the RIC discovers and sets up a trusted connection with all the target gNBs and APs. The user then provides control functions requirements to the matching agent. The agent then generates the CFR document with the matched gNB or AP control functions. This CFR document is then sent to gNB and AP. This information is received by the codegen agents on each NFs. These agents generate the code required for creating the interface API server. Note that the API server is unique for every vendor. The agent tests the code until the correct version is generated. Once this is successful it then sends a provisioning complete message back to RIC. The RIC then triggers the interface client to start sending control messages.

V. Implementation and Performance

In order to test our use case, we use a custom multi-RAT RIC implementation with an embedded matching agent as shown in Figure 3. We simulate gNB and WLAN AP using containers. As commercial vendors do not divulge their internal API, we created a set of custom internal APIs for both gNB and AP which can trigger simulated control functions. For example, *getRateStats* and *releaseUE*. Our API simulates 30 control actions each for gNB and AP. In order to simulate a multivendor scenario, we created multiple sets of APIs that are functionally similar but semantically different (e.g. varying function names, data types and units). We test using 5 simulated multivendor gNBs and 5 simulated multivendor APs.

For both agents, we use a RAG pipeline in combination with two LLMs: GPT-4o and Llama3.3-70B. The GPT-4o model runs on OpenAI servers, while the Llama3.3 model runs on a server with Nvidia Tesla V100 (128GB VRAM). We use bge-small-en-v1.5 as the embedding model with Llama3.3 and text-embedding-ada-002 as an embedding model for GPT-4o.

The RAG pipeline consists of three stages- document indexing, query-based retrieval, and response generation. In the document indexing stage, the input text document is tokenized to create embeddings. These embeddings are stored in a vector database for later retrieval. During the query-based retrieval stage, the vector database is searched for text embeddings most relevant to the LLM input query. The top 3 most relevant embeddings are fetched and are passed to the LLM. In the response generation phase, the LLM generates a reply to the input query based on the retrieved text embeddings. In case of the matching agent, the NF capability document and the control function requirements from user are used as input documents for the RAG pipeline. For the codegen agent, the CFR document along with vendor API documentation is used as input for its RAG pipeline. We use custom prompts that are tuned specifically to the functionality of the agents.

A. Matching Agent performance

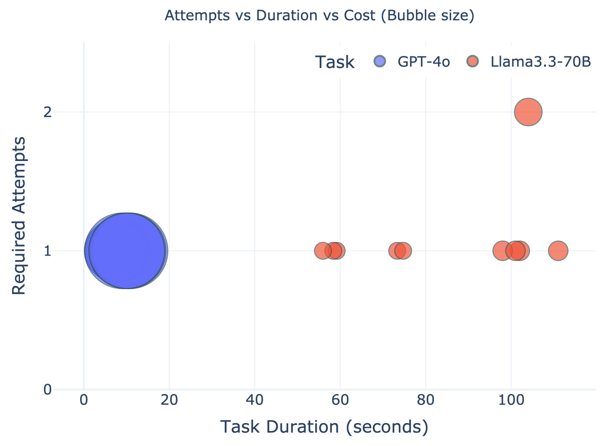

Fig. 4. Matching Agent performance

To test the performance of the matching agent, we created 60 custom control function requirements based on the capability of AP/gNB. To account for the diverse nature of user input requirements, we generated ten distinct variations of these requirements using Claude 3.5 Sonnet [9]. We then used these variations as input to the matching agent and observed whether it correctly matches them to the capabilities of the AP/gNB and how many attempts were required to achieve a correct match. Figure 4 shows the number of attempts and the total time required to match 60 requirements. The diameter of the circle represents the cost. We observe that GPT-4o successfully matches all requirements on the first attempt. Llama3.3 achieves the same but with a few exceptions. Moreover, GPT-4o is 6 to 10 times quicker than Llama3.3, depending on the complexity of the matching task. However, GPT4-o is approximately 14 times more costly than Llama3.3.

B. Codegen agent performance

To evaluate the performance of the codegen agent, we attempt to generate correct and executable code based on the CFR document provided by the matching agent. We generate code for 5 APs and 5 gNBs, each using distinct simulated

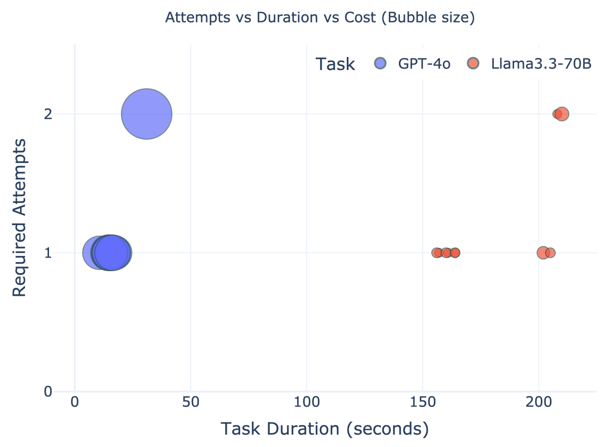

Fig. 5. Codegen agent performance

vendor APIs. Figure 5 shows the number of attempts and the total time required to generate the code while the diameter of the circle represents the cost. We observe that the code generation was predominantly successful on the first attempt. Compared to the matching task, coding task is more complex. Here, the performance gap between Llama3.3 and GPT-4o increases significantly. In this case, GPT-4o based agent is 10-14 times faster compared to the Llama3.3 based agent but also costs 14 times more.

Besides the above experiments, we also tried to generate augmented control functions based on the existing NF capabilities. This approach is used when the control function requirement from the user does not have an exact match with the capabilities of the NF but a similar functionality is available. We tested two control functions:

- Age of Information (AoI) aware rate control: This control function is designed to set the specified data rate on the AP, provided that the given timestamp has not already elapsed. The rate setting is not applied if the given timestamp exceeds the AP's system time. In this case, the AP vendor supports rate control but lacks AoI aware rate control. The codegen agent is expected to add a wrapper on top of vendor API which can compare the provided timestamp against system time and generate the API server.
- Timestamped telemetry collection: In this case, the control function requires all the returned data to be timestamped. The gNB vendor supports data collection but does not include timestamps. The codegen agent is tasked with writing a wrapper around vendor API which can read system time and append it to returned data.

We discovered that the codegen agent was able to successfully augment the functions when clear and structured instructions were provided in the requirements. On average, this task took 3 times longer and was 3 times more expensive than generating non-augmented control functions. However, when we tried to create more complex augmented control functions involving generating multi-threaded code, the agent was unable to produce the correct code despite multiple attempts. This could only be resolved with manual intervention which involved modifying the original prompt.

C. *Overall results*

Overall, we observed that GPT4-o offered the best E2E performance. On average, generating an interface with 10 control functions costs \$0.04 and has an E2E latency of 9.4s. Furthermore, GPT-4 showed minimal variation in latency for a given agent with varying task complexity, whereas Llama3 shows a large variation in latency, even for similar kind of tasks.

VI. Limitations

We highlight some of the limitations of our work below:

- We conduct all of our experiments on synthetic data. This is due to the fact that most vendors do not expose their device API. Although we aim to model our synthetic data (control function requirements, vendor APIs) as realistically as possible, performance in real-world scenarios may vary significantly, necessitating further validation under diverse operational conditions. The performance of agents also depends on the availability of well-structured and detailed capability and API documentation [10].
- Considering real-world deployment of this framework, security of network functions might be affected by LLM specific threat vectors. These might be direct or indirect prompt injection or context windows overloading [11]. In our experiments, we observed that the size of the context window matters for accuracy. If the number of control functions in a single query is too large the models start to produce inaccurate output [12]. However, this can be addressed by splitting it into multiple smaller queries.
- In the case of matching agent, there is no ground truth against which the agent can validate the LLM model output. Hence, we had to validate the output by using another LLM for evaluation. This can be extended to use a set of specialized LLMs to evaluate the output which, however, also increases the cost. Moreover, this method may not be fully trustworthy due to the inherent uncertainty of LLMs. In real-world deployments, the generated interface might need to be tested in a sandbox before deploying in production.
- Due to the virtualization of the network edge, it is possible to deploy large AI models in the edge network functions. However, these models might be too big for devices such as WLAN APs. Although this could be mitigated by using cloud based models over an API, we believe that in the future the models would get small enough to be deployed on devices like APs.

VII. Related work

NetconfEval [12] introduces a model agnostic benchmark to evaluate LLMs for network configuration. It evaluates multiple use cases, developing routing algorithms, generating low level network configuration, converting high level requirements into formal specifications and function calls.

Netllmbench [13] introduces a framework for benchmarking LLMs in network configuration tasks. It compares various LLMs including Llama3-70B in terms of their speed of operation and number of iterations required.

NAIL [14] discusses a method to translate high-level instructions from the user into executable code for P4 using transpiler. It also has ability to continuously monitor the network to ensure intent fulfillment.

The system COSYNTH [15] uses verified prompt programming for generating router configurations. They demonstrate its application by translating Cisco router configurations into equivalent Juniper router configurations.

In [16], the authors discuss the integration and compatibility challenges in a multi-vendor O-RAN deployment. They propose a framework using WebAssembly plugins to enable interoperable and flexible multi-vendor deployments.

Zero-touch Service Management (ZSM) [17] frameworks can integrate AI/ML models to autonomously detect and resolve anomalies within 6G networks.

VIII. Conclusion and Future Work

We present a novel framework to create network control interfaces on demand between any two set of NFs. We achieve this by leveraging LLMs to develop a multi-agent framework. Our framework brings flexibility and interoperability to the control interfaces. We believe that in future mobile networks, AI agents might be used natively in all parts of the network [18]. Generating control interface on demand using AI is a step in this direction.

In this work, we only focus on control interfaces and not on the applications that make the control decisions (e.g., xApps). We believe the decision-making applications can also be generated using AI in the future. We plan to investigate how generated control applications and interfaces could be used to create autonomous control loops for 6G networks.

Acknowledgment

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany (BMBF) in the program of "Souverän. Digital. Vernetzt." Joint project 6G-RIC (16KISK020K). This project was also supported by German Federal Ministry for Digital and Transport funded project 5G-COMPASS (19OI22017A).

References

[1] "3gpp 23.501," 2025, accessed: 2025-01-02. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-j20.zip

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[4] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *The Knowledge Engineering Review*, vol. 10, no. 2, p. 115–152, 1995.

[5] M. Polese, L. Bonati, S. D'oro, S. Basagni, and T. Melodia, "Understanding o-ran: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 1376–1411, 2023.

[6] "Ric platform - confluence/wiki," 2024, accessed: 2024-12-22. [Online]. Available: https://lf-o-ran-sc.atlassian.net/wiki/spaces/RICP/overview?homepageId=14123010

[7] "mosaic5g / flexric Â· gitlab," https://gitlab.eurecom.fr/mosaic5g/flexric, 2024, accessed: 2024-12-22.

[8] P. D. Thapa, A. Kappen, and J. Schulz-Zander, "Towards infrastructure-assisted wifi rate adaptation for converged networks with morpheus," ser. MobiArch '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3691555.3696828

[9] "Claude 3.5 sonnet," https://www.anthropic.com/news/claude-3-5-sonnet, 2024.

[10] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. Del Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, and OTHERS, "Textbooks are all you need," 2023. [Online]. Available: http://arxiv.org/pdf/2306.11644

[11] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 1831–1847.

[12] C. Wang, M. Scazzariello, A. Farshin, S. Ferlin, D. Kostić, and M. Chiesa, "Netconfeval: Can llms facilitate network configuration?" *Proc. ACM Netw.*, vol. 2, no. CoNEXT2, Jun. 2024. [Online]. Available: https://doi.org/10.1145/3656296

[13] K. Aykurt, A. Blenk, and W. Kellerer, "Netllmbench: A benchmark framework for large language models in network configuration tasks," in *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2024, pp. 1–6.

[14] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "Nail: A network management architecture for deploying intent into programmable switches," *IEEE Communications Magazine*, vol. 62, no. 6, pp. 28–34, 2024.

[15] R. Mondal, A. Tang, R. Beckett, T. Millstein, and G. Varghese, "What do llms need to synthesize correct router configurations?" in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, ser. HotNets '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 189–195. [Online]. Available: https://doi.org/10.1145/3626111.3628194

[16] R. Cannatà, H. Sun, D. M. Dumitriu, and H. Hassanieh, "Towards seamless 5g open-ran integration with webassembly," in *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, ser. HotNets '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 121–131. [Online]. Available: https://doi.org/10.1145/3696348.3696864

[17] M. Liyanage, Q.-V. Pham, K. Dev, S. Bhattacharya, P. K. R. Maddikunta, T. R. Gadekallu, and G. Yenduri, "A survey on zero touch network and service management (zsm) for 5g and beyond networks," *Journal of Network and Computer Applications*, vol. 203, p. 103362, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804522000297

[18] W. Tong, "A-ran, a-core, a-ue: The bridge to 6g," https://www.eucnc.eu/wp-content/uploads/2024/06/Keynote-Wen-Tong.pdf, 2025, accessed: 2025-01-09.