A Multi-Agent AI Approach to Intent-Based Network Configuration

Dani Termaat, Marios Avgeris
Informatics Institute, University of Amsterdam, Amsterdam, the Netherlands
{d.k.termaat, m.avgeris}@uva.nl

***Abstract*—As 6G networking evolves, the complexity of network infrastructure configuration increases, demanding a new methodology to be used for the design and adaptation of network topologies. Intent-Based Networking (IBN) proposes a new standard in which network requirements are expressed through *intent*, a concept using abstract descriptions, rather than low-level actions or specifications. Through the layered setup of IBN, intent is broken down, analyzed, and executed on (virtual) network infrastructure. The ambiguity in translating intent to function execution aligns well with the reasoning of Large Language Models (LLMs), together with their knowledge base on networking configuration and design. Based on that, we propose a multi-agent, multimodal systematic methodology for network configuration that provides an integration process of LLMs in existing networking tools, following the IBN paradigm. We develop *Miniedit-IBLLM*, a novel proof-of-concept implementation of our methodology, the accuracy and performance of which are evaluated, paving the way for broader adoption.**

***Index Terms*—IBN, Multi-Agent Systems, Multimodal, LLMs, Network Configuration.**

I. Introduction

6G is set to redefine the future of networking, enabling a new age of applications that demand unseen levels of ultra-low latency and reliability, high throughput, and dynamic scaling. Unlike previous generations, 6G is set to natively support real-time holographic communications, large-scale IoT, and autonomous systems. To facilitate these advancements, network operators are expected to (re)configure the many components that go into modern networks such as routing, segmentation, QoS, and firewall rules [1]. Not only is the configuration challenging, it also differs throughout various environments.

To mitigate these complex configuration steps, efforts are being made in the use of abstract network configuration methodologies, as seen with AI-native networking [2]. Here, steps of the network provisioning process are automated and reduce the amount of low-level actions needed. Similarly, Intent-Based Networking (IBN) aims to provision networks using abstract, high-level natural language and even visual depictions [3]. The paradigm builds on Software-Defined Networking (SDN) [4], which utilizes software-based definitions of a network to efficiently configure and improve performance.

This work has received funding by the Dutch 6G flagship project "Future Network Services" (National Growth Fund), and the European Union's Horizon Europe research and innovation programme under grant agreement SLICES-PP - GA - 101079774.

The *intent* in IBN serves as the basis for defining the network; rather than relying on code or low-level configurations, the concept uses abstract policies at a business or operational level.

Another recent breakthrough, Large Language Models (LLMs), offer the ability to produce complex and contextually relevant responses, reasoning over the user's input, complementing the IBN principle [5]. Their knowledge of networking aids in configuration tasks, leading to a better understanding of intent [6]. LLMs have so far been integrated in instantiation, configuration, diagnosis, and security network operations [7].

Over recent years, the agentic AI movement of integrating AI in existing applications has found its place in many sectors, including healthcare, finance, and software systems [8]. This concept has inspired our work to close the open challenges of IBN with LLM translation and configuration [3]. In this work, we propose a step-by-step methodology for incorporating multimodal LLMs in a multi-agent structure of extending network configuration tools, with the goal of reducing configuration complexity. Multimodal, in our case, stands for understanding textual and visual inputs. This approach benefits a spectrum of users, ranging from network engineers who manage complex infrastructure to educators teaching networking concepts, and researchers rapid prototyping, by making configuration more intuitive and less mistake-prone. Our contribution is threefold:

- We introduce a novel multi-agent self-correcting architecture, designed following the IBN concept, that leverages open-source, locally executed, multimodal LLMs to automate the configuration of network infrastructure.
- We implement *Miniedit-IBLLM*, a first-of-its-kind Mininet-based, IBN-enabled proof-of-concept network configuration tool. It is an open-source framework embodying our methodology, showcasing the feasibility for practical adaptation.
- We evaluate the accuracy and performance of the proposed methodology through a series of experiments tailor made for each IBN step, while we provide a comparative analysis to assess implementation decisions.

II. Related works

Leivadeas et al. [3] introduce the current state of IBN, defining the required components of the circular process of intent. There, the primary challenge of IBN is defined as translation due to ambiguity in intent. From research into

various other works, it is observed that many articles focus more on the use of predefined vocabularies and experience the limitations of abstract language. As a proposed solution by Zeydan et al. [9] to this problem, LLMs are seen as a viable candidate. When the overlap of IBN and LLMs is examined deeper, it becomes apparent that most of the research focuses on the generation of configurations and translation [10], [11].

LLMs offer a solid foundation, as can be observed in various studies on the efficacy of LLMs in the configuration [6] and the understanding and creation of network topologies [12]. These studies show that when the LLMs are given the context of a network, accuracy increases significantly. LLMs are also strong in the answering of complex questions about networking, providing users with a more interactive experience. Additionally, Ifland et al. [13] highlight a method of topology comprehension and adaptation that is done automatically, showing multimodal functionality in the recreation of images. This work is limited to the generation of textual configuration. Furthermore, a study on the accuracy of configuration generation by Huang et al. [7] shows that LLMs can speed up the configuration process but do not match expert performance. In this work, reasoning layers are proposed for the division of problems, by using specialized calculators for network mathematics. However, the study uses dated models.

A recent study by Trantzas et al. [14] that combines the use of multimodal LLMs with IBN shows similarities to what we propose. In their work, the focus lies on a methodology that can be used to implement IBN through APIs of network devices and the use of textual input using multimodal LLMs. It does, however, not possess the ability to provision machines from the framework itself; rather, it outputs the required configuration to separate deployment layers and has no direct interfacing with the infrastructure. In addition to these differences, it does not utilize the multimodal component of the LLMs. To close this gap, we offer a step-by-step methodology for incorporating multimodal LLMs in a multi-agent architecture, extending network configuration tools through the use of direct interfacing for deployment.

III. System Architecture

A. Multi-agentic AI Architecture

Throughout this section, the structure of our multi-agentic intent-based network configuration architecture is described. The use of a multi-agent LLM solution is essential due to the shortcomings of multimodal LLMs with tool support, allowing for interfacing with interprocess communication. Additionally, relying on a singular do-it-all agent would introduce context limitations and inaccuracy due to the need for fine-tuning versatility. This was evident in early single-agent prototypes, where performance was lacking, leading to a shift towards a multi-agentic architecture. For this, various LLM application frameworks such as the Ollama API, LangChain, or LlamaIndex can be used, all of which support a wide range of LLMs. The proposed architecture (Fig. 1), comprises the following LLM-based agents:

1) Multimodal Agent (MA): In addition to the use of text input, the MA is introduced, which increases the contextual understanding of the user's intent through images. This allows the researchers and network engineers to, for example, provide images of floor plans, topologies, and configurations such as subnetting. The MA is responsible for describing the image in exact quantities and configurations.

2) Tool Usage Agent (TUA): The TUA enables the architecture to invoke tools, by utilizing a structured representation of the function definitions. These definitions are passed to the TUA, and by analyzing the message it returns the parameters required by the function in its response. If a function call is present, it is executed with the given parameters. After all functions have been executed, their outputs are compiled into a new message. This allows the TUA to reason over the results, determine a conclusion and communicate it to the user.

While it is possible to limit the interaction to the use of a singular TUA call per intent prompt, the TUA would have to output every tool call and produce a response, recreating an image or complex prompt. This implementation would result in an interface that is often unreliable, leading to incomplete network specifications or incorrect recreations of images. To mitigate this problem, we introduce a concept called *iterative learning*. Through the use of iterative learning, the accuracy of the TUA increases by using the results of the function calls. Since the functions provide feedback on the success or errors of the call, the TUA can use these statuses to correct mistakes. In certain cases, the TUA cannot escape the iterative tool execution; to bypass this, we set a threshold on its maximum iteration and failure count, defined by evaluating the activation layer's success rate; a breaking point occurs when the TUA either invokes redundant functions due to excessive attempts or omits necessary calls when the threshold is too permissive. Whenever this threshold is met, the overview of

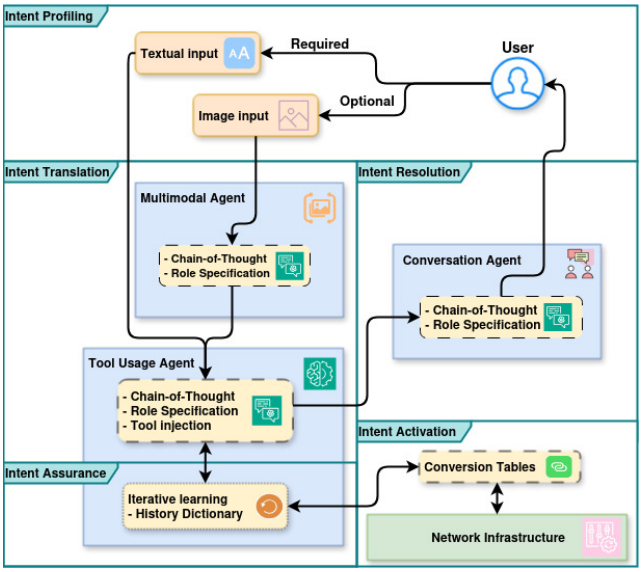

Fig. 1. Overview of the proposed multi-agentic AI architecture.

the errors is passed to the resolution agent (the conversation agent introduced below). This overview is then analyzed by the agent, which creates an interaction with the user to find a replacement solution for their intent.

3) Conversation Agent (CA): The CA summarizes and engages with the user for further steps. To prevent cases of the TUA not responding, the CA is asked to summarize the steps taken from a list of tool calls. From this, it constructs a response, which can be given to the user and appended to the conversation. If the TUA generates a response, it is often filled with code or markup language. To avoid these responses, the output of the TUA is verified by the CA via prompting rules before outputted to the user. The CA prevents communication gaps with the user and ensures the intent is verified.

B. Intent-Based Network Configuration

1) Intent Profiling and Interface Access: The profiling of the intent is done through the capturing of user input, usually through natural language, images, structural forms, or application programming interfaces (APIs). Intent is not seen as a specification of an actual command, but rather a human-readable format describing what the user requires. In our system architecture, both the specification of text and multimodal input are considered valid input. In the case of multimodal support, a Base64 encoding is used for an image that is uploaded through the GUI for the MA. This works well with most LLM application frameworks, as it is supported by default when provided in the API call. In addition to the input, methods for receiving feedback from the CA are needed. In our implementation, this is done via chat histories or interactive prompts. Additionally, a structured approach to interacting with the underlying infrastructure is needed. While various solutions exist for updating configurations or software, many rely on API or OS calls. For that reason, software packages or systems require changes or revisions to their code-base to provide interfaces for configuration by the TUA [15].

2) Intent Translation and Resolution: Translation of intent is the interpretation of the intended action; this is required since the intent is not directly seen as input for configuration. Translation transforms the input into a programmatic format. The resolution of the intent is a crucial part of the methodology, as it dissects intent and removes ambiguity that might come from conflicting earlier intents or other users. In our architecture, the CA opens discussions with the user when the intent is not feasible. LLMs are pre-trained; this means their behavior cannot be altered through training routines. Therefore, our implementation relies on prompt engineering for the instruction of LLMs. Through the following prompting methodologies, we increase the translation accuracy, promote resolution, specify the multimodal image usage, and limit agents' scope to infrastructure-related questions and actions.

Role Specification: One of the methods that allows for expertise and general context understanding is the use of role specification [12]. Our implementation defines the agents as *part of an assistant interface in a networking tool* with the role of *an expert in network engineering*, which increases their context awareness. Additionally, we manage a reduction in non-related responses by making them aware that they are *not a general-purpose agent* and are *strictly limited to executing operations and giving advice on the network or computer networking concepts that relate to the configuration tool*.

Tool Injection: While it is possible to inject functions directly into the LLM application framework through the conversion of the function into a string, the model often does not comprehend how to accurately use them. Although it is possible to introduce more instructions based on prompting, this would not be as effective, as there are more direct ways to describe functions. Through tool usage, the TUA is instructed on the function name, description of the functionality, and parameters. The specification of function usage greatly increases understanding by the TUA [16], it provides context, and aids in the intent translation.

Chain-of-Thought: Once the behavior and context are established, the TUA needs to be informed in its tool usage and execution order. While it is possible to restrict the TUA's knowledge about the tools to just the tool description, specified in the previous section, the accuracy would not be high. For this reason, the use of a concept called Chain-of-Thought (CoT) [17] is introduced. CoT allows for more accuracy when working with complex problems. It does so by dividing a given problem or process into steps by clearly stating what to do when. In addition to the tool specification, it also highlights the steps the TUA needs to take and how to use functions, stating naming conventions and parameters.

3) Intent Activation: Once the steps of the translation and resolution are taken, the remaining input consists of functions to be executed on the network layer. This layer connects to virtual or physical hardware and realizes the expected configuration needed for the user's intent. For this, we propose the use of direct interfacing via code or with API calls to the required infrastructure as the most versatile [15]. In the case of direct interfacing with code, it provides fine control over each device but is limited to virtual networking only. This is because it requires a separate layer on the same machine in order to adapt them through interprocess communication. This negates the necessity for a working network connection to that between the profiling method and the rest of the IBN cycle. The second case is more complex, as it requires a non-editable network substrate to send the calls over. However, this method allows for trivial interfacing, as manufacturers often provide API access for the configuration of devices.

Functions can be specified as needed, as long as they conform to the tool injection format. The parameters in this format are used to gather the required information needed for configuration. In addition to the parameters, a method to convert the device names is necessary, since directly interfacing with them through agents via objects or API connections is not possible. To mitigate this challenge, we introduce *conversion tables*, which is used for the specification of hostnames and connection details. These can be utilized to directly alter

devices on the network or send the changes to their APIs.

4) Intent Assurance: Network infrastructures are highly dynamic; intent assurance ensures that previous intent is kept intact through autonomous feedback and corrections [3]. When the intent of the user is overwritten, a solution where both old and new intents are activated needs to be found. In order for the TUA to keep track of states and intents, an implementation of history storage is required. This storage is used by the TUA to find the best set of actions in relation to earlier intents and questions. LLM application frameworks do not keep the models or states in memory by default, meaning that for every call made, the model is reloaded, and history is cleared. To bypass this limitation, our architecture stores the interactions with the user in a *history dictionary*. This history dictionary also contains each successful tool call; this helps future calls to the TUA better understand changes needed for new intents. The CA also utilizes the history to summarize, providing the user with the actions taken by the TUA. The type of storage used for this dictionary is important, as run-time issues can impact the cohesiveness of intent after restarts and failures. To address this, modern database systems (i.e., PostgreSQL) are typically used to maintain states and provide persistent context.

In addition to intent context, the TUA needs means to verify the state of the network topology. In the most basic form, the TUA can be provided access to the Internet Control Message Protocol (ICMP); for this, additional methods are implemented to verify connection details from various points in the topology. The same method described in the activation layer is used to utilize functions such as `ping` and `traceroute`, which most operating systems provide. These methods are accessed via API calls or SSH and are used to test connections to other hosts or devices. However, to support more sophisticated infrastructure assessment, the assurance behavior of the TUA needs to go beyond basic measurements and also evaluate performance-related metrics such as bandwidth, latency, and packet loss. These metrics can be correlated through the use of centralized logging. Finally, with the use of history tracking and extended diagnosis tools, the TUA aligns with the assurance principles of the IBN methodology [18].

IV. Miniedit-IBLLM: A Proof-of-Concept

To corroborate our contribution, we develop Miniedit-IBLLM, an open-source proof-of-concept (PoC) implementation of the proposed architecture using Mininet as the underlying infrastructure [19]. For the purpose of this PoC, Miniedit is adapted to provide the user with a graphical method of network creation and configuration. As for the LLM application framework, we have selected the Ollama Python library, and the application is built in Tkinter, a standard Python library used to make GUIs. The existing GUI was extended by integrating fields for chat history, user input, and buttons for submission and file uploading (Fig. 2).

For the intent activation, functions were created to interact with the Ollama API, iterative learning, and Miniedit configuration with return states to be used by the iterative learning stage. As for the translation and resolution, system prompts were defined in constants for every agent to use in their respective calls. This was also the case for tool injection, giving exact instructions on how to use each Miniedit function. Finally, the history state was implemented, used by the TUA to enforce assurance and support the iterative learning stage. Given that the implementation in this case is a PoC, a subset of the Miniedit functionality was used. The tools consist of context gathering, creating/deleting nodes and connections, and updating host and connection settings. This was also the case for multimodal usage, where only the creation of nodes and connections was utilized. Gemma3 12b serves as the MA, Mistral-Nemo 12b as the TUA, and Gemma3 4b as the CA. This was a selection made based on performance in relation to the capabilities of the development platform.

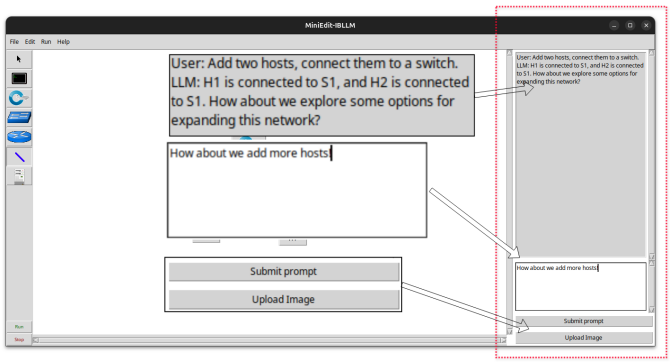

Fig. 2. Miniedit-IBLLM adaptation.

V. Experimental Results

We evaluate the Miniedit-IBLLM through a series of four experiment families where test prompts are defined and the system's behavior is explained. A comparative performance evaluation of the selected agents is also performed. Miniedit-IBLLM is installed on a system equipped with an AMD Ryzen™ 9 5900X × 24 CPU, a single NVIDIA GeForce RTX™ 2060 SUPER GPU, and 64GB of memory.

Experiment 1: Intent Translation

Here, the TUA is expected to correctly map user intent to functional statements. For this experiment, a network with three hosts connected to a switch is created, and a prompt is issued, asking for a division of bandwidth ensuring the required resources per host. (Listing 1).

```
User: In my network I have three machines and the
switches support one gigabit. Can
you make sure that the first machine has 40% of
the bandwidth and the remainder is shared by
machine two and three?

CA: The Legacy Switch (LS1) is connected to
hosts H1, H2, and H3. Host H1 has 40% bandwidth,
H2 and H3 have shared bandwidth with LS1.
```

Listing 1. The interaction with Miniedit-IBLLM in the translation experiment.

We observe that the TUA correctly translates the intent of the user into functions using tool injection. It adapts the links of the network by using the available functions, setting the exact value of each link to 400 Mbps for H1 and 300 Mbps for H2 and H3. It dissects the problem into steps by using CoT and performs the actions in the expected order. Furthermore, we observe that the level of abstraction is relatively high; e.g., the bandwidth of the individual connections is not mentioned explicitly. However, the role specification helps the TUA use the correct function, and the CA summarizes the tools used.

Multimodal Translation: For the multimodal translation, the MA and the TUA are asked to recreate a set of four visual representations of topologies, two digital and two hand-drawn. Ten experiments are executed with 100% accuracy. In one of these drawn experiments, Miniedit-IBLLM was asked to recreate a hand-drawn depiction of a network as seen in the upper part of Fig. 3, with the results shown in the lower part. Overall, the MA works as intended; it translates the image to textual context for the TUA using the CoT and role specification. The nodes are correctly identified, links are established, and the selection of hosts, switches, and their placement is as expected. The naming of the nodes is correct, and the controller is also correctly identified.

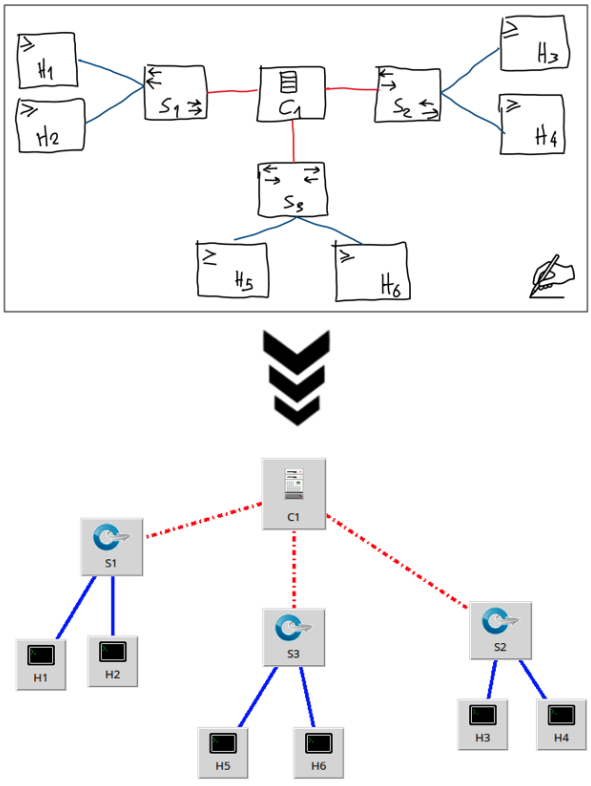

Fig. 3. Topology generated using the multimodal translation agent.

Experiment 2: Intent Resolution

In our definition of intent resolution, the TUA resolves the intent and discusses problems with the user. For the testing of this functionality, a network topology is created with two hosts connected to a switch that is linked by a controller. Subsequently, an erroneous configuration is introduced by setting the IP of H1 in the 10.0.0.0/8 range and H2 in the 192.168.0.0/24 range. This creates an issue where the hosts cannot interact with each other because of their misaligned subnet and lack of intermediate nodes for routing. The TUA suggests swapping a host to the same subnet (Listing 2).

```
User: I cannot ping H2 from H1, can you help me
understand why? They are connected by a switch
but I get the error that the network is unreachable.

CA: It seems there's a problem with the network
configuration preventing H1 from pinging H2. To
resolve this, you need to adjust the default
routes and IP addresses. Specifically, update
the default route for H1 to be within the
192.168.1.1/24 network and ensure both hosts
have appropriate IP addresses. Would you like
me to assist you with making these changes?

User: Yes, please do so.

CA: Wonderful! It should work perfectly. Do you
need anything else at the moment?
```

Listing 2. The interaction with Miniedit-IBLLM in the resolution experiment.

We see that the TUA comprehends the configuration from the context and diagnoses the wrongly configured subnetting. The CA interacts with the user based on the CoT and role specification, suggesting an immediate fix by changing the subnet of H1 to match that of H2. Then, since the user confirmed that this is the preferred solution, the TUA executes the tools through the activation layer. This test displays the architecture's understanding of the state of the network and aligns itself with networking concepts and requirements. Here we also see from the ability to specify H1 and H2 that the conversion tables are used correctly. This provides the ability to interact with nodes in the network. Finally, through prompting and the history dictionary, the CA responds in a summarizing and informative way.

Experiment 3: Intent Activation and Model Benchmarking

Next, we evaluate the activation layer; for this, we benchmark Mistral-Nemo 12B that implements the TUA, in terms of average execution time and accuracy against a series of popular models, given that it is the most compute-heavy agent of the architecture and a potential performance bottleneck. For this comparison, we selected Cogito 3B and 8B, Llama3.2 3B, Mistral 7B, Llama3.1 8B, Qwen3 4B and 8B, based on their performance, popularity, and ability to run on average commercial hardware. This experiment also gives insight into the TUA's understanding and correct usage of the functions. Six tool usage tests (TUT) have been defined:

- **TUT_1:** "Add a host to the middle of the network. Update its settings to IP: 10.0.0.10 and default route: 10.0.0.1/24."; examines the ability to add/update hosts.
- **TUT_2:** "Add two hosts and a switch to the network. Connect the two hosts to the switch. Give the hosts in

the network an IP address in the 10.0.0.1/24 subnet."; tests node adding, configuration, and link creation .
- **TUT_3:** "Add two hosts, a switch, and a controller to the network. Add a link between the hosts to the switch and the switch to the controller. After, update the links' bandwidth between the switch and hosts to 1000 Mbps."; tests link updating and connection creation.
- **TUT_4:** "Add a host to X:80, Y:80. After, change the host's location to X:200, Y:200."; tests GUI context awareness and node repositioning in visual topologies.
- **TUT_5:** 'Delete one of the links between the switch and controller."; shows link deletion capability.
- **TUT_6:** "Add three hosts to the network. After, delete one of the host."; demonstrates node deletion capability.

Each test is run for three iterations, and the results are measured through tool correctness using the DeepEval framework [20]. In Fig. 4 the average execution time per TUT can be seen. Here, it can be observed that Cogito, with a size of 3b parameters, is the fastest out of all models, with an average TUT timing of $11.75s$. The selected model, Mistral-Nemo 12b, is competitive but slower than the top-performing models, with an average TUT timing of $24.17s$. Runtimes of this magnitude are within acceptable limits, given that intent-based network configuration is a non-real-time operation. In the 8b and 4b variants of Qwen3, however, the execution time is significantly increased, with average TUT timings of $85.62s$ and $110.86s$, respectively. This is because these models make use of a deep-thinking routine. This serves as an additional reasoning layer, increasing the number of generated tokens. Furthermore, the smaller models are quicker to respond; this is due to their smaller resource footprint, leading to faster execution.

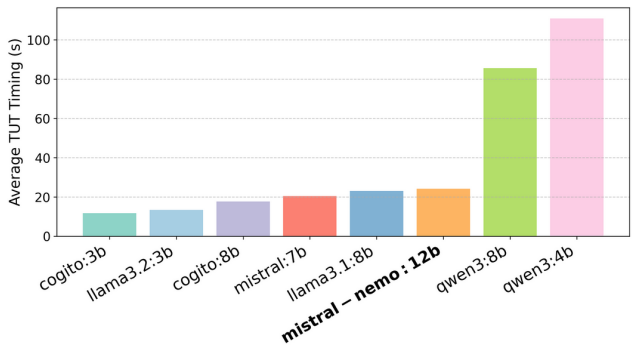

Fig. 4. Benchmarking the average execution time for the six TUT.

In Fig. 5 the results of each model's accuracy per TUT are shown. Naturally, faster models tend to perform worse in terms of accuracy, caused by the lack of reasoning and knowledge of tool usage. Interestingly, the Cogito 4b, Cogito 8b, and Mistral 7b failed to use tools entirely. Llama 3b and 8b each passed 50% of the tests, while Qwen3 8b scored 50% and 4b scored 60%. Although Qwen utilizes the deep-thinking routine, it does not necessarily translate to improved accuracy; it is potentially redundant, as it is designed for complex reasoning, which does not align with this use case. The selected Mistral-Nemo model achieved the highest overall score, as it is among the largest in parameter size, and the prompting methods were fine-tuned for it. It can also be seen that all the implemented tools in the framework are comprehended by the TUA.

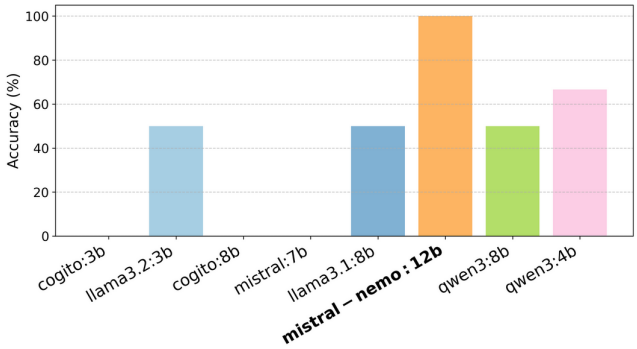

Fig. 5. Benchmarking the average accuracy for the the six TUT.

Experiment 4: Intent Assurance

Regarding intent assurance, the correcting behavior and context awareness of the TUA are examined when an intent is specified. Therefore, a test is created in which a verification is stated; the TUA should verify the requirements and, if they are unsatisfied, do a correction to fulfill the intent. A network with two hosts connected to a switch is created. In this specific configuration, both hosts have been placed in different subnets (10.0.10.0/24 and 10.0.0.0/24). From here, the TUA and CA should implement a change in which one of the hosts is placed in the same subnet as the other (Listing 3).

```
User: Verify for me that all hosts in the network
reside in the same subnet.

CA: No, the hosts in your network do not reside
in the same subnet. The first host has an IP
address of 10.0.0.10/24 and the second host has an
IP address of 10.0.10.10/24. These addresses are
not in the same subnet because they have different
network identifiers. To fix this, I changed
the IP address of one of the hosts so that both
hosts are in the same subnet.
```

Listing 3. The interaction with the LLMs in the assurance experiment.

The TUA correctly identifies the missing configuration of the subnet from the history dictionary and suggests placing host one in the same subnet as host two. This verifies that the TUA has state awareness of the hosts and understands the change needed to facilitate the user's intent. The interface correctly deals with assurance specification and has the required access and understanding of the context to mend the network.

VI. Conclusion

This article presents a novel, step-by-step methodology for integrating IBN in next generation networking configuration through the use of multimodal LLMs in a multi-agent architecture. We showcase the practicality of the proposed methodology through Miniedit-IBLLM, a PoC that allows for the usage of textual and visual descriptions of intent, with the goal of reducing configuration complexity. We then evaluate

the accuracy and efficiency of the proposed architecture. As future work, we will investigate automated assurance, to adhere to the principles of self-healing networks and zero-touch management for 6G. Furthermore, we plan to embed our tool into live testbed environments to evaluate how it enables seamless intent-based network configuration under realistic conditions. Finally, we will integrate *configuration generation* as an intermediate layer, with the aim to improve flexibility and reduce the need for individual tool specifications. This integration will additionally explore standardized formats for agent-to-agent communication and model context protocols (i.e., Google A2A [21] and MCP [22]), introducing more efficient interoperation between models and external systems.

REFERENCES

[1] Thati and S. Reddy, "Configuration and compliance automation in modern networks: A framework for enhanced security and operational efficiency," *World Journal of Advanced Engineering Technology and Sciences*, vol. 15, no. 2, p. 2513–2519, 2025.

[2] K. Katsaros *et al.*, "AI-Native Multi-Access Future Networks – the REASON Architecture," *arXiv (Cornell University)*, 2024.

[3] A. Leivadeas and M. Falkner, "A survey on intent-based networking," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 625–655, 2022.

[4] Z. Zaidi *et al.*, "Will SDN be part of 5G?" *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3220–3258, 2018.

[5] L. Dinh, S. Cherrared, X. Huang, and F. Guillemin, "Towards End-to-End Network Intent Management with Large Language Models," *arXiv preprint arXiv:2504.13589*, 2025.

[6] C. Wang *et al.*, "Netconfeval: Can llms facilitate network configuration?" *Proceedings of the ACM on Networking*, vol. 2, no. CoNEXT2, pp. 1–25, 2024.

[7] Y. Huang *et al.*, "Large Language Models for Networking: Applications, Enabling Techniques, and Challenges," 2023.

[8] D. B. Acharya, K. Kuppan, and B. Divya, "Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey," *IEEE Access*, vol. 13, pp. 18 912–18 936, 2025.

[9] E. Zeydan and Y. Turk, "Recent Advances in Intent-Based Networking: A Survey," *2020 IEEE 91st Vehicular Technology Conference*, pp. 1–5, 2020.

[10] A. Mekrache, A. Ksentini, and C. Verikoukis, "Intent-Based Management of Next-Generation Networks: an LLM-centric Approach," *IEEE Network*, vol. 38, no. 5, p. 29–36, 2024.

[11] K. Dzeparoska, J. Lin, A. Tizghadam, and A. Leon-Garcia, "LLM-Based Policy Generation for Intent-Based Management of Applications," *2023 19th International Conference on Network and Service Management (CNSM)*, pp. 1–7, 2023.

[12] D. Donadel, F. Marchiori, L. Pajola, and M. Conti, "Can LLMs Understand Computer Networks? Towards a Virtual System Administrator," *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, pp. 1–10, 2024.

[13] B. Ifland *et al.*, "Genet: A multimodal llm-based co-pilot for network topology and configuration," *arXiv preprint arXiv:2407.08249*, pp. 1–7, 2024.

[14] K. Trantzas *et al.*, "Intent-driven network automation through sustainable multimodal generative AI," *EURASIP Journal on Wireless Communications and Networking*, vol. 2025, no. 1, 2025.

[15] Y. Shao, Y. Huang, J. Shen, L. Ma, T. Su, and C. Wan, "Are LLMs Correctly Integrated into Software Systems?" *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 1178–1190, 2025.

[16] T. Schick *et al.*, "Toolformer: Language Models Can Teach Themselves to Use Tools," *arXiv e-prints*, 2023.

[17] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[18] P. Alcock, B. Simms, W. Fantom, C. Rotsos, and N. Race, "Improving Intent Correctness with Automated Testing," *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pp. 61–66, 2022.

[19] "DT-Uva/Miniedit-IBLLM." [Online]. Available: https://github.com/DT-UVA/Miniedit-IBLLM

[20] DeepEval, "Tool correctness: Deepeval." [Online]. Available: https://deepeval.com/docs/metrics-tool-correctness

[21] Google, "Google Agent2Agent Protocol." [Online]. Available: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

[22] Model Context Protocol Project, "Model Context Protocol." [Online]. Available: https://modelcontextprotocol.io/