Specification and Evaluation of Multi-Agent LLM Systems - Prototype and Cybersecurity Applications

Felix Härer
University of Applied Sciences Northwestern Switzerland
Basel, Switzerland
felix.haerer@fhnw.ch

***Abstract*—Recent advancements in LLMs indicate potential for novel applications, e.g., through reasoning capabilities in the latest OpenAI and DeepSeek models. For applying these models in specific domains beyond text generation, LLM-based multi-agent approaches can be utilized that solve complex tasks by combining reasoning techniques, code generation, and software execution. Applications might utilize these capabilities and the knowledge of specialized LLM agents. However, while many evaluations are performed on LLMs, reasoning techniques, and applications individually, their joint specification and combined application is not explored well. Defined specifications for multi-agent LLM systems are required to explore their potential and their suitability for specific applications, allowing for systematic evaluations of LLMs, reasoning techniques, and related aspects. This paper reports the results of exploratory research to specify and evaluate these aspects through a multi-agent system. The system architecture and prototype are extended from previous research and a specification is introduced for multi-agent systems. Test cases involving cybersecurity tasks indicate feasibility of the architecture and evaluation approach. In particular, the results show the evaluation of question answering, server security, and network security tasks that were completed correctly by agents with LLMs from OpenAI and DeepSeek.**

***Index Terms*—LLM, Multi-Agent System, Reasoning, Cybersecurity.**

I. Introduction

With the release of recent LLMs such as the DeepSeek R and OpenAI o variants, LLMs have demonstrated advancements in terms of reasoning capabilities [1], [2]. Well-known problems and benchmarks for advanced mathematical challenges such as AIME2024 were successfully tackled [3], [4], although they had been set out only recently, through novel prompting and reasoning techniques, e.g., by constructing reasoning chains of prompts and responses.

The recent advancements on models and techniques clearly demonstrate progress on a technological level and, yet, their actual implementations in real-world applications are only beginning to become clear. While there does not seem to be a shortage of services and apps offering various interfaces to generative AI, the high-impact applications remain to be identified and assessed systematically.

For instance, systematic evaluations need to determine whether specific LLMs are suitable and how they could be utilized for applications in various domains, how the LLMs and techniques compare in specific applications, in addition to advanced comparisons such as the coordination of LLM agents issuing prompt inputs and reacting to outputs for different interaction patterns, techniques, and LLMs such as established models and new reasoning models. At this point, systematic evaluations such as benchmarks concern LLMs, techniques, and related aspects individually and allow for individual comparisons.

In order to explore the potential to combine the specialized knowledge and capabilities of LLM agents with reasoning and prompting techniques in specific applications, these LLM agent specifications in multi-agent LLM systems require further exploration in order to assess the potential in applications and for systematic evaluations on an application level. This paper reports on the initial results of experimental research towards evaluating multi-agent LLM systems through prototyping. The resulting artifact is an LLM execution system, initiated in 2023 [5], that has been extended for specifying and evaluating multi-agent LLM applications. In particular, an extended architecture and a defined specification support multi-agent systems that can combine multiple, specialized LLMs and support prompting and reasoning techniques in the execution of tasks. A system overview is presented in this paper, consisting of the architecture and specification language together with test cases for cybersecurity tasks. The test cases evaluate network and server security tasks with agent specifications using commercially and openly available state-of-the-art LLMs by OpenAI and DeepSeek.

In the remainder of this paper, Section II introduces background and related work, the architecture and specification are discussed in Section III, and demonstrated with cybersecurity test cases in Section IV. Section V concludes.

II. Background and Related Work

Agents can carry out actions to complete complex tasks, individually or jointly in the loosely coupled structures of multi-agent systems [6], [7]. In principle, agents act on their own and commonly involve functional components depending on the environment, e.g., agents using sensors in industrial environments or software agents with executable functions and corresponding data. Generally, tasks are orchestrated in the agent system by interaction among agents such as invoking specialized agents with specific functions, knowledge, or data. In the context of Large Language Models (LLMs), these models serve as the foundation for completing tasks [8], [9],

using prompts and responses that invoke functions, read or write data, or perform system-specific actions for completing tasks and solving problems.

LLMs based on Generative Pre-trained Transformers (GPT) predict sequences of tokens that are mapped to entities such as words or word fragments, pixel or related image and video feature representations, and other data entities [10]–[12]. These values are encoded and embedded to be represented in a high-dimensional vector space, in addition to the positions in the sequence. Passing through multiple layers of the GPT architecture, in the form of transformer blocks, attention and feed-forward network components predict tokens and probabilities based on attention values [13] and sampling for the selection of any following token. Since attention relates to prior positions in the sequence with similar attention, these positions represent and find related concepts; rather than using only probabilities to predict the next token. For this reason, the semantic capabilities of agents have recently been greatly enhanced as evidenced by recent model releases.

Recently, OpenAI and DeepSeek demonstrated reasoning models, where the network components are trained by reinforcement-learning together with prompting and reasoning techniques to produce chains of logically following reasoning steps [1]. Reasoning models gain the ability to produce one or multiple subsequent answers, which might be arranged as a tree or graph, and to evaluate the answers in a subsequent reasoning step. By considering one or multiple answers together with the context of previous steps, this input allows the LLM to evaluate coherence and select answers to continue accordingly. In tree or graph structures, multiple paths might be explored before selecting a coherent line of reasoning [14], [15]. Thus, reasoning models allow for reflection and also introspection, e.g., visible the openly available DeepSeek models that generate a "¡think¿" tag for reasoning that is closed before the final answer is returned.

Reasoning chains and techniques such as Chain-of-Thought (CoT) can achieve behavior similar to reasoning and have also demonstrated recent advancements [16], [17]. CoT is a well known approach that issues prompts in order to generate the responses that reflect on the original question and the conversation context, thus, it is not limited to reasoning models. Several variants and further techniques exist. Notably, augmentation of prompts, e.g., using retrieval augmented generation, can add information or knowledge relevant to a prompt and improve answers [18]. Formulating constraints with zero- or few-shot prompting [14], [16], [19] allows the LLM to learn from examples that are concrete example or provide abstract guidance or structures. Constraints might also be placed on the content of answers, where related aspects are required to adhere to defined criteria, being less prone to hallucinations. In case structural or syntactical properties are constrained, the replication of the structure or syntax tends to be replicated and instantiated correctly by the LLM.

While these techniques can be applied generally, they are not considered explicitly in LLM or agent specifications, e.g., for platforms or evaluations of LLMs and agents. Several domain-specific benchmarks exist for comparing LLMs such as for Cybersecurity applications [20]–[22] or sub-disciplines such as threat intelligence [23]. The results generally indicate high accuracy for larger, state-of-the-art models and point out the future potential in the area. This view is shared also by two recent surveys [24], [25] that provide a comprehensive overview.

While multi-agent LLM systems are beginning to appear in practice, the specification of multi-agent LLM systems is still not understood well and underexplored. Especially the specification by a format or language that allows for constraints, reasoning chains, and related prompting and reasoning techniques with multiple agents is not evident in literature. Thus, the specification and platform, demonstrated by this explorative research paper, is meant to enhance the understanding of multi-agent systems and inform future LLM responses and agent development by systematic evaluations. E.g., by evaluating reasoning techniques, LLMs designed for reasoning or not designed for reasoning, and combining techniques selectively depending on capabilities and knowledge.

III. Multi-Agent LLM System Architecture and Specification

The following subsections discuss the overall concept based on high-level requirements, a system architecture realizing the requirements, and a specification for multi-agent LLM systems.

A. High-Level Requirements

In order to outline the concept of a multi-agent LLM system, the following high-level requirements are defined to establish the architecture:

1. Specification of the access to open source and commercial LLMs through interfaces and parameters with an API or a local LLM runtime such as Ollama.
2. Specification of agents invoking LLMs with prompts, task actions and other agents with behavior depending on LLM outputs, action execution results, results of other agents, and the evaluation of results.
   - 2.1 Specifying LLM prompts in the conversation contexts of one or multiple agents in data structures that support prompting and reasoning techniques in terms of sequential prompts, multiple chains, and constraints.
   - 2.2 Specifying task actions such as executable commands together with data defined at build-time and variably set within agents at run-time. The action execution inputs and outputs as well as data must be inserted into the conversation contexts.
   - 2.3 Evaluating results at run-time, originating from LLM responses, outputs of action executions, or other agents, i.e., the evaluation through other LLM agents. An evaluation must conclude whether a result satisfies defined criteria, e.g., computation or calculation results falling in defined classes or reaching an expected value.

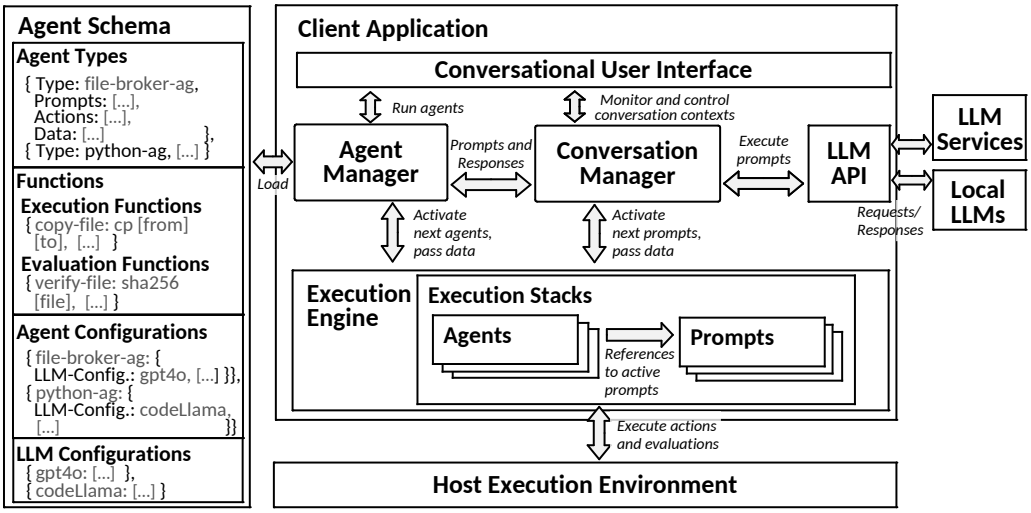

Fig. 1: Architecture describing the components of the client application together with the specification in the agent schema.

2.4 Invoking prompts, action execution for an agent or other agents at run-time unconditionally or conditionally based on previous results and evaluations.

The requirements related to prompting and evaluations concern in multi-agent systems the capability to support specific techniques such as chaining or constraining the reasoning in the context of action executions. For example, constraints can be placed on prompts and responses regarding their format, e.g., text strings, integer or decimal values, patterns, or complex syntax rules of formal languages or programming languages. In this case, the evaluation encompasses a syntax evaluation of the LLM responses before execution by, e.g., a python-capable agent.

B. Execution Architecture

The following architecture extends previous research on LLM execution [5] to a multi-agent system architecture with a corresponding specification, realizing the high-level requirements. Architecture and specification are shown in Figure 1 and in the form of a client application with its sub-systems and an agent schema for specifying agent types with related functions and configurations.

The sub-systems manage agents and the conversation through the components *Agent Manager*, *Conversation Manager* and *Conversational UI*. Given an agent schema, *Agent Manager* requests user interface (UI) actions in *Conversational UI* such as agent settings and the loading of prompts in *Conversation Manager*. When running an agent, *Conversation Manager* activates the agent in the *Execution Engine* by pushing it onto a stack for execution together with the referenced prompts, activated by the *Conversation Manager*. An active prompt is executed by *Conversation Manager* through *LLM API* while loading prompts and streaming responses to *Conversational UI* for monitoring and conversation control. The architecture and prototype support external LLM services with common APIs such as the OpenAI API and Replicate in addition to local LLM applications such as Ollama. Active agents reference their next prompts, both placed on separate stacks. The stack-based execution calls for any prompt at the top of the stack In the stack-based execution, any prompt at the top of the stack is (1.) processed by an LLM via *Conversation Manager*, (2.) the generated response is received by the topmost agent, and (3.) the agent executes all specified functions on the *Host Execution Environment* (HEE) and supplies the response as input. The functions referenced in a prompt are defined in the form of execution or evaluation functions, where execution functions relate to actions that are executed first, e.g., a ”copy-file” function of the agent file-broker-ag, running in an Ubuntu Server HEE. After executing all actions, the specified evaluation functions such as ”verify-file” will be called and determine the result of the execution. For example, such a function might compute a file checksum using sha256 and match the result with an expected value, resulting in a true evaluation result in case of a match or in false evaluation otherwise. Further examples are discussed in Section IV.

C. Agent Schema Specification

For specifying multiple agents, executable tasks, and evaluation functionality, a schema-based approach is realized, drawing from conceptual modeling and object orientation. On an abstract schema level, the schema provides a template for agent types and related functions and configurations. The specification in Figure 1 shows an overview of the domain-specific language structure with examples. In the prototype, the concrete syntax is realized in JSON format.

An *Agent Schema* encompasses *Agent Types*, *Functions*, *Agent Configurations*, and *LLM Configurations*. Agents are instantiated based on *Agent Types* that specify the executable prompts and actions together with data in the form of key-value data structure. Actions specify the behavior of agent types, defining the actions an agent can carry out to complete tasks. They reference *Execution Functions* that consist of executable commands with inputs and outputs denoted in data variables and *Evaluation Functions* for the evaluation

```
[ { "id": "Security-Q&A-Agent",
    "prompts": [ {
        "prompt": "In TCP/IP networking, which protocol is used to hold network addresses and
            routing information in a packet?",
        "answers": { "A": "HTTP", "B": "IP", ... },
        "answer": "B" }, ... ],
    "prompt-template": "Question: [question]\n\nOptions:\nA) [answers/A]\nB) [answers/B]\nC) [
        answers/C], ...",
    "evaluate": {
        "result-classes": [ {
        "class": "A", "pattern": "ANSWER: A", "eval-expected": "correct", "eval-unexpected": "
            incorrect" }, ... ] } },
  { "id": "Network-Security-Agent",
    "prompts": [ {
        "prompt": "Scan the local network [ipv4-network] for reachable hosts with commonly
            exposed ports. Use the nmap module.",
        "actions": ["write-to-file", "extract-ip-scan-results"],
            "expected-value": "10.11.1.24" }, ... ],
    "actions": ["write-to-file", "extract-code", "evaluate-syntax-shell", "execute-shell"],
    "data": {
        "report-file": "network-report.txt", "ipv4-network": "10.1.1.0/24" }, ... } ]
```

Listing 1: Agent-Type specification for question answering in a Security Q&A Agent and for network scanning tasks in a Network Security Agent. Based on questions and answers from the CyberMetric dataset [20], performance is evaluated for the LLM and the configuration of the Q&A agent. For task execution, performance is evaluated based on expected classes of results, defined by values or patterns. E.g., the Network Security Agent is expected to find open ports at host 10.11.1.24.

of results produced by an agent. At runtime, function inputs are the LLM response and the data, held in the key-value data structure in the agent on the stack. Functions read and write to the data structure, e.g., they might extract code blocks from an LLM response, evaluate the syntax of generated python code, or execute arbitrary code. *Evaluation Functions* operate on the LLM response and execution outputs. They specify conditions, values, patterns, and classes to evaluate an expected or unexpected result. The action specification on the agent level applies to all prompts and may be overwritten on the prompt level. There, it may be specified unconditionally or in a case statement conditionally, in case the denoted result class applies.

Furthermore, *Agent Configurations* specify configurations required to initialize each agent with its defined type at runtime, including function parameters, data, and a reference to an LLM configuration defined in *LLM configurations*, where LLM and API parameters are set.

IV. Test Cases for Cybersecurity Applications

This section discusses exemplary test cases with an evaluation for cybersecurity tasks involving an agent specification with OpenAI and DeepSeek models. The test cases encompass question answering as well as server and network security tasks that apply prompting and reasoning techniques. These test cases aim at evaluating the feasibility of the architecture and specification.

A. Q&A Specification by Templating

Answering a set of questions with the expectation of open or pre-defined answers is a foundational application for LLMs. The LLM is presented with a series of questions, where each response is evaluated against an expected result by an evaluation function, e.g., determining a match with a value or pattern individually or in pre-defined classes. Case 1 applies this concept in a series of 10 cybersecurity questions from the CyberMetric Q&A dataset [20] as an example. Listing 1 shows the definition of an agent type Security Q&A Agent that is prompted with a Q&A specification. A template defines the format for questions and answers and specifies loading questions and answers from a file or inline. Evaluation functions are defined by classes A to D, matching the expected results in the dataset.

According to the specification, the evaluation computes result classes by agent and prompt. As shown in Table I, results indicate all LLMs managed to complete the 10 test questions correctly, indicating basic cybersecurity knowledge and basic question answering capabilities. The source code and the complete results are published online[1].

B. Task Execution Specification

For executing tasks, their individual actions are specified for an execution environment. Case 2 sets up server and network security agents with 8 tasks aimed at typical activities, including assessing firewall configurations and scanning a local network in Ubuntu Linux 24.04. Listing 1 shows the type Network Security Agent with instructions to generate a network scanning script, executed according to the defined actions. Further instructions on using Python in the Ubuntu environment were given in the system-prompt. The python

[1]https://github.com/fhaer/multi-agent-llm-system

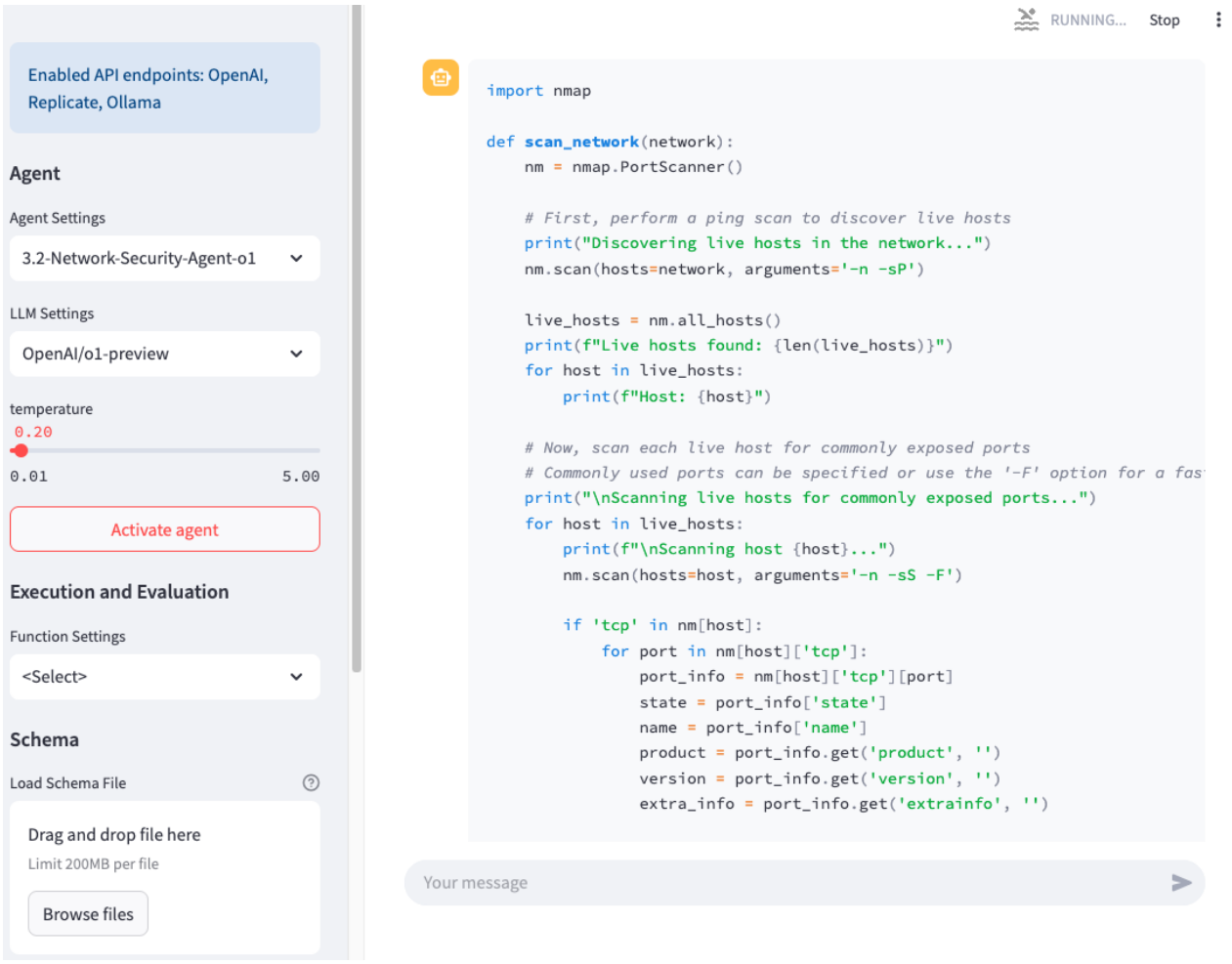

Fig. 2: Excerpt of a Python script generated for the network scanning task in Listing 1 within the user interface of the client application. The code was generated and executed by a python-capable agent of the type Network Security Agent.

code generated and executed by the network security agent is shown in Figure 2.

1) Augmentation and Constraints: in the processing of prompts and responses are used for task executions. Prompts are augmented at run-time with agent data or results of previous executions as specified by agent data keys, e.g., ipv4-network. In this way, retrieval augmented generation (RAG) is possible also for information or knowledge retrieval, e.g., by actions containing database or knowledge graph queries. Constraints reduce the solution space and narrow possible answers by specifying aspects of the response structure or content. When executing actions, the constraints specified in prompts are enforced, e.g., requesting and requiring a response in a specific data type such as an IPv4 address, a language such as python, or data format such as JSON.

Here, the agent executes generated Python scripts according by actions that check the syntax, execute, and further process results. Thus, the agent-type must rely on an LLM with sufficient domain knowledge, programming capabilities for python, and the Ubuntu 24.04 environment. For the server and network security tasks, Table I shows the evaluation results.

The tested LLMs are state-of-the-art models at the time of this writing, except for a relatively small DeepSeek-R1-Distilled LLM (R1D) with only 8 billion parameters that was added for comparison. All of the large state-of-the-art models

TABLE I: Results of correct and incorrect task executions by the specified agents. Agent IDs denote the agent LLMs OpenAI GPT-4o (4o), OpenAI o1-preview (o1), DeepSeek R1 (R1), and DeepSeek R1 Distilled Llama 8B (R1D).

ID	Correct Tasks	Incorrect Tasks	Total Tasks
1.1-Security-Q&A-Agent-4o	10	0	10
1.2-Security-Q&A-Agent-o1	10	0	10
1.3-Security-Q&A-Agent-R1	10	0	10
1.4-Security-Q&A-Agent-R1D	10	0	10
2.1-Server-Security-Agent-4o	4	0	4
2.2-Server-Security-Agent-o1	4	0	4
2.3-Server-Security-Agent-R1	4	0	4
2.4-Server-Security-Agent-R1D	2	2	4
3.1-Network-Security-Agent-4o	4	0	4
3.2-Network-Security-Agent-o1	4	0	4
3.3-Network-Security-Agent-R1	4	0	4
3.4-Network-Security-Agent-R1D	4	0	4

```
[ { "id": "Server-Security-Agent",
    "prompts": [ {
       "prompt": "...",
       "invoke": {
           "agent-of-type": "Audit-Report-Agent",
           "prompt-id": 1,
           "data-keys": ["ipv4-address", "ipv4-network", "scan-result"] } }, ... ] },
  { "id": "Audit-Report-Agent",
    "prompts": [ {
       "prompt": "Create a report of the findings for the server with IP address [ipv4-address]
           in the network [ipv4-network]. Consider each of the hosts found in the scan result,
           indentify potentially vulnerable services, and give recommendations to address
          potential vulnerabilities. \n\nScan result:\n\n[scan-result]", ... } ], ... } ]
```

Listing 2: Agent-Type specification, where a Server Security Agent invokes an Audit Report Agent and passes data.

completed each of the 4 network and server security tasks correctly. R1D managed to complete all 4 network security tasks, and 2 of the 4 server security tasks correctly. In the incorrect cases, R1D constructed an incorrect shell command for retrieving the system firewall configuration (server security task 2) and used an incomplete command for creating a system report (server security task 4). The complete results are published online.

2) Reasoning Chains: can be constructed based on a series of prompts and responses together with constraints [14], [16]. In addition to Chain-of-Thought (CoT), which aims to issue prompts that generate subsequent reasoning steps with their responses, the chain can also be constructed explicitly by relying on pre-defined structures. Constraints ensure their existence and allow prompting the LLM of an agent with the next step. This is especially relevant in multi-agent scenarios, where another, specialized agent might be invoked based on previously collected data. For example, Listing 2 shows the Server-Security-Agent type that establishes an IP address, network, and scan result data after a series of prompts and invokes an agent of type Audit-Report-Agent.

On the basis of these prompts, the agent is prompted to create a report using the passed data. Further actions might place constraints or specific conditions influencing which agent is invoked or the data passed to the agent. Figure 3 in Appendix A shows an excerpt of the generated report.

V. Conclusion

This paper presented the initial results of experimental research on the specification of multi-agent LLM systems. In this first system overview, the architecture and specification language were demonstrated and applied in test cases related to question answering as well as network and server security tasks. The test cases indicate feasibility of the architecture and specification, showing the completion of tasks with agent specifications based on state-of-the-art commercially and openly available LLMs. In particular, the completion of the test cases shows the potential of LLM agents for software-based tasks in applications. Factors enabling the task completion are a combination of (1.) involving the knowledge capabilities of specialized agents such as for cybersecurity and code generation, (2.) utilizing reasoning techniques such as reasoning chains, and (3.) agents executing the inferred tasks through software actions. In future research, this combination will allow the exploration of novel multi-agent LLM applications with advanced reasoning requirements and, overall, support systematic evaluations in cybersecurity and other domains.

References

[1] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, and et al., "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," 2025, arXiv:2501.12948 [cs].

[2] J. A. Pruet, K. Duraisamy, V. Agrawal, A. Biswas, R. B. Bujack, M. J. Grosskopf, A. A. Hagberg, B. Hu, E. C. Lawrence, W. Li *et al.*, "Implications of new reasoning capabilities for science and security: Results from a quick initial study," Los Alamos National Laboratory (LANL), Los Alamos, NM, US, Tech. Rep., 2024.

[3] AoPS Incorporated, "2024 AIME II," 2024, https://artofproblemsolving.com/wiki/index.php/2024_AIME_II, Retrieved 2025-02-23.

[4] Y. Brett, "OpenAI Introduces o3," 2024, In: Weights & Biases. https://wandb.ai/byyoung3/ml-news/reports/OpenAI-Introduces-o3-Pushing-the-Boundaries-of-AI-Reasoning–VmlldzoxMDY3OTUxMA, Retrieved 2025-02-23.

[5] F. Härer, "Conceptual model interpreter for Large Language Models," in *ER Forum 2023, 42nd International Conference on Conceptual Modeling (ER 2023)*, vol. 3618. CEUR-WS, 2023.

[6] D. Maldonado, E. Cruz, J. Abad Torres, P. J. Cruz, and S. d. P. Gamboa Benitez, "Multi-Agent Systems: A Survey About Its Components, Framework and Workflow," *IEEE Access*, vol. 12, pp. 80 950–80 975, 2024.

[7] Y. Li and C. Tan, "A survey of the consensus for multi-agent systems," *Systems Science & Control Engineering*, vol. 7, no. 1, pp. 468–482, 2019.

[8] X. Dong, X. Zhang, W. Bu, D. Zhang, and F. Cao, "A Survey of LLM-based Agents: Theories, Technologies, Applications and Suggestions," in *2024 3rd International Conference on Artificial Intelligence, Internet of Things and Cloud Computing Technology (AIoTC)*, 2024.

[9] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large Language Model Based Multi-agents: A Survey of Progress and Challenges," in *Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024*, vol. 9, 2024, pp. 8048–8057.

[10] P. Kumar, "Large language models (LLMs): survey, technical frameworks, and future challenges," *Artificial Intelligence Review*, vol. 57, no. 10, p. 260, 2024.

[11] P. P. Ray, "Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 121–154, 2023.

[12] D. Rothman, *Transformers for Natural Language Processing. Second Edition*. Birmingham, UK: Packt Publishing, O'Reilly Media, 2022.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Red Hook, NY, USA, 2017.

[14] L. Luo, Z. Zhao, C. Gong, G. Haffari, and S. Pan, "Graph-constrained Reasoning: Faithful Reasoning on Knowledge Graphs with Large Language Models," 2024, arXiv:2410.13080 [cs].

[15] A. Prasad, S. Saha, X. Zhou, and M. Bansal, "ReCEval: Evaluating Reasoning Chains via Correctness and Informativeness," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, 2023, pp. 10 066–10 086.

[16] Z. Lin, C. Chan, Y. Song, and X. Liu, "Constrained Reasoning Chains for Enhancing Theory-of-Mind in Large Language Models," in *PRICAI 2024: Trends in Artificial Intelligence*. Singapore: Springer Nature, 2024, pp. 354–360.

[17] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-Consistency Improves Chain of Thought Reasoning in Language Models," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

[18] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, and Z. Liu, "Evaluation of retrieval-augmented generation: A survey," in *Big Data*. Singapore: Springer Nature Singapore, 2025, pp. 102–120.

[19] H.-G. Fill, F. Härer, I. Vasic, D. Borcard, B. Reitemeyer, F. Muff, S. Curty, and M. Bühlmann, "CMAG: A Framework for Conceptual Model Augmented Generative Artificial Intelligence," in *ER Forum 2024, 43rd International Conference on Conceptual Modeling (ER 2024)*, vol. 3849. CEUR-WS, 2024.

[20] N. Tihanyi, M. A. Ferrag, R. Jain, T. Bisztray, and M. Debbah, "CyberMetric: A Benchmark Dataset based on Retrieval-Augmented Generation for Evaluating LLMs in Cybersecurity Knowledge," in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, London, UK, 2024, pp. 296–302.

[21] D. Bhusal, M. T. Alam, L. Nguyen, A. Mahara, Z. Lightcap, R. Frazier, R. Fieblinger, G. L. Torales, B. A. Blakely, and N. Rastogi, "SECURE: Benchmarking Large Language Models for Cybersecurity," in *Proceedings of the 40th Annual Computer Security Applications Conference*, ser. ACSAC '24. New York, NY, USA: Association for Computing Machinery, 2024.

[22] Z. Liu, J. Shi, and J. F. Buford, "CyberBench: A Multi-Task Benchmark for Evaluating Large Language Models in Cybersecurity," in *The AAAI-24 Workshop on Artificial Intelligence for Cyber Security (AICS)*, Vancouver, CA, 2024.

[23] M. T. Alam, D. Bhusal, L. Nguyen, and N. Rastogi, "CTIBench: A Benchmark for Evaluating LLMs in Cyber Threat Intelligence," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, CA*, 2024.

[24] J. Zhang, H. Bu, H. Wen, Y. Liu, H. Fei, R. Xi, L. Li, Y. Yang, H. Zhu, and D. Meng, "When LLMs meet cybersecurity: a systematic literature review," *Cybersecurity*, vol. 8, no. 1, p. 55, 2025.

[25] S. Sai, U. Yashvardhan, V. Chamola, and B. Sikdar, "Generative AI for Cyber Security: Analyzing the Potential of ChatGPT, DALL-E, and Other Models for Enhancing the Security Space," *IEEE Access*, vol. 12, pp. 53 497–53 516, 2024.

APPENDIX A

Appendix A contains the result of generating an audit report by an agent of the type Audit Report Agent, which was previously invoked by an agent of the type Network Security Agent. The full results are published in the online repository at https://github.com/fhaer/multi-agent-llm-system.

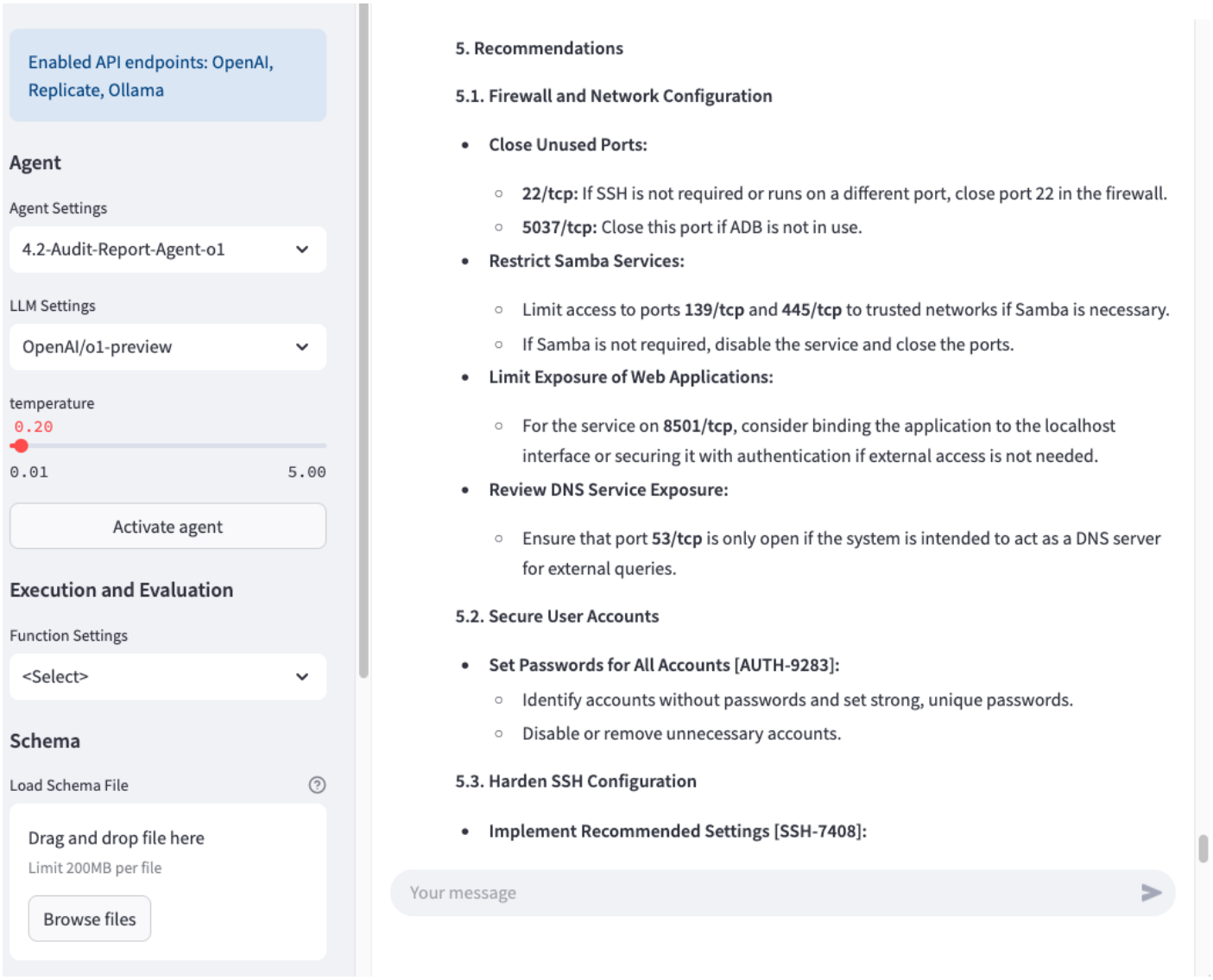

Fig. 3: Excerpt of a security report generated by an Audit Report Agent after the completion of server-side tasks by a Network Security Agent in the user interface of the client application.