ETSI GR ENI 051 V4.1.1 (2025-02)

GROUP REPORT

Experiential Networked Intelligence (ENI); Study on AI Agents based Next-generation Network Slicing

Disclaimer

The present document has been produced and approved by the Experiential Networked Intelligence (ENI) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference

DGR/ENI-0051v411_AI-Agent_NGCN

Keywords

6G, GenAI, LLM, native AI

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Important notice

The present document can be downloaded from the
ETSI Search & Browse Standards application.

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format on ETSI deliver repository.

Users should be aware that the present document may be revised or have its status changed,
this information is available in the Milestones listing.

If you find errors in the present document, please send your comments to
the relevant service listed under Committee Support Staff.

If you find a security vulnerability in the present document, please report it through our
Coordinated Vulnerability Disclosure (CVD) program.

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.
No recommendation as to products and services or vendors is made or should be implied.
No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.
In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2025.
All rights reserved.

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Experiential Networked Intelligence (ENI).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document analyses and studies potential ENI activities on 6G native AI capabilities. ENI has a strong focus on providing cognitive capabilities to improve the user experience of the operator. The present document covers the areas and needs for new technical projects using 6G native AI capabilities. This covers: definition of the use cases and requirements of Core Network (CN) Large Language Models (LLMs), definition of the CN-Agent to facilitate interaction between the CN and the CN LLMs, identification of the key functional modules and interfaces of the CN-Agent, and the key technologies required. Network slicing is used throughout the present document to explain the operation of this system. However, this is not a limiting case, and the system described in the present document is intended to serve a large variety of CN use cases.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] 3GPP TS 23.288 (V16.0.0): "Architecture enhancements for 5G System (5GS) to support network data analytics services (Release 16)".

[i.2] 3GPP TS 23.288 (V17.0.0): "Architecture enhancements for 5G System (5GS) to support network data analytics services (Release 17)".

[i.3] 3GPP TS 23.501 (V19.0.0): "System architecture for the 5G System (5GS); Stage 2 (Release 19)".

[i.4] ETSI GR ENI 010 (V1.2.1): "Experiential Networked Intelligence (ENI); Evaluation of categories for AI application to Networks".

[i.5] ETSI GS ENI 030 (V4.1.1): "Experiential Networked Intelligence (ENI); Transformer Architecture for Policy Translation".

[i.6] J. Ruan, Y. Chen, B. Zhang, et al.: "Tptu: Task planning and tool usage of large language model-based ai agents", NeurIPS 2023 Foundation Models for Decision Making Workshop, 2023.

[i.7] H. Pan, Z. Zhai, H. Yuan, Y. Lv, et al.: "KwaiAgents: Generalized Information-seeking Agent System with Large Language Models", arXiv:2312.04889, 2024.

[i.8] C. Li, H. Chen, M. Yan, W. Shen, et al.: "ModelScope-Agent: Building Your Customizable Agent System with Open-source Large Language Models", arXiv:2309.00986, 2023.

[i.9] Chen W, Su Y, Zuo J, et al.: "Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors". The Twelfth International Conference on Learning Representations. 2023.

[i.10] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch: "Improving factuality and reasoning in language models through multiagent debate", CoRR, abs/2305.14325, 2023. doi:10.48550/arXiv.2305.14325.

[i.11] Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji: "Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona selfcollaboration", CoRR, abs/2307.05300, 2023b. doi: 10.48550/arXiv.2307.05300.

[i.12] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan: "Building cooperative embodied agents modularly with large language models", CoRR, abs/2307.02485, 2023a. doi: 10.48550/arXiv.2307.02485.

[i.13] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu: "Chateval: Towards better llm-based evaluators through multi-agent debate", 2023.

[i.14] J. Wei, X. Wang, D. Schuurmans et al.: "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models", Advances in Neural Information Processing Systems (NeurIPS), 2022.

[i.15] X. Wang, J. Wei, D. Schuurmans et al.: "Self-consistency improves chain of thought reasoning in language models", arXiv: 22203.11171, 2022.

[i.16] J. Long: "Large Language Model Guided Tree-of-Thought", arXiv: 2305.08291, 2023.

[i.17] M. Besta, N. Blach, A. Kubicek, et al.: "Graph of Thoughts: Solving Elaborate Problems with Large Language Models", Proceedings of the AAAI Conference on Artificial Intelligence, 2024.

[i.18] S. Yao, J. Zhao, D. Yu, et al.: "ReAct: Synergizing Reasoning and Acting in Language Models", International Conference on Learning Representations (ICLR), 2023.

[i.19] Yan M., Agarwal S., Venkataraman S.: "Decoding speculative decoding". arXiv preprint arXiv:2402.01528, 2024.

[i.20] Leviathan Y., Kalman M., Matias Y.: "Fast inference from transformers via speculative decoding". International Conference on Machine Learning. PMLR, 2023: 19274-19286.

[i.21] Chen C., Borgeaud S., Irving G., et al.: "Accelerating large language model decoding with speculative sampling". arXiv preprint arXiv:2302.01318, 2023.

[i.22] Liu X., Hu L., Bailis P., et al.: "Online speculative decoding". arXiv preprint arXiv:2310.07177, 2023.

[i.23] X. Liu, H. Yu, H. Zhang, et al.: "AgentBench: Evaluating LLMs as Agents", arXiv:2308.03688, 2023.

[i.24] J. Lin, H. Zhao, A. Zhang et al.: "AgentSims: An Open-Source Sandbox for Large Language Model Evaluation", arXiv: 2308.04026, 2023.

[i.25] J. Lu, T. Holleis, Y. Zhang et al.: "ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities", arXiv: arXiv:2408.04682, 2024.

[i.26] Y. Shavit, S. Agarwal, M. Brundage et al.: "Practices for governing agentic AI systems", Research Paper, OpenAI, December, 2023.

[i.27] NGMN Alliance (V1.0): "6G Use Cases and Analysis".

[i.28] Recommendation ITU-R M.2160-0 (V1.0): "Framework and overall objectives of the future development of IMT for 2030 and beyond".

[i.29] ETSI GS ENI 005 (V3.1.1): "Experiential Networked Intelligence (ENI); System Architecture".

[i.30] Q. Wu, G. Bansal, J. Zhang, et al.: "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation", Conference on Language Modeling (COLM), August 2024.

[i.31] O. Ayan, S. Hirche, A. Ephremides, W. Kellerer: "Optimal Finite Horizon Scheduling of Wireless Networked Control Systems", IEEE/ACM Transactions on Networking, April 2024.

[i.32] ETSI GR ENI 016 (V2.1.1): "Experiential Networked Intelligence (ENI); Functional Concepts for Modular System Operation".

[i.33] ETSI GR ENI 007 (V1.1.1): "Experiential Networked Intelligence (ENI); ENI Definition of Categories for AI Application to Networks".

[i.34] D. López-Pérez, A. De Domenico, N. Piovesan and M. Debbah: "Data-Driven Energy Efficiency Modeling in Large-Scale Networks: An Expert Knowledge and ML-Based Approach", in IEEE Transactions on Machine Learning in Communications and Networking, 2024.

[i.35] ETSI TR 121 905: "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Vocabulary for 3GPP Specifications; (3GPP TR 21.905)".

[i.36] ETSI GS ENI 019 (V3.1.1): "Experiential Networked Intelligence (ENI); Representing, Inferring, and Proving Knowledge in ENI", 2023.

[i.37] EI AI Act Regulation (EU) 2024/1689 of the European Parliament and of the European Council.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

AgentGPT: domain-specific model trained with domain-specific knowledge that matches the responsibilities of the AI Agent that it resides in

AI Agent: autonomous system that can interact with its environment to collect data, learn from the past experiences and subsequently use these to improve its decision-making capability in order to perform specific tasks

NOTE: As defined in clause 4.2.1 also [i.6], [i.7] and [i.8].

E2E Slice: logical network that provides a combination of specific network and network capabilities and network characteristics, supporting various service properties for network slice customers

E2E Slice Instance: set of Network Function, and Application Function instances and the required computing and communication resources that form a deployed E2E Slice

functional block: abstraction that defines a black box structural representation of the capabilities and functionality of a component or module, and its relationships with other functional blocks

intent policy: type of policy that uses statements from a restricted natural language (e.g. an external DSL) to express the goals of the policy, but does not specify how to accomplish those goals

NOTE: As defined in [i.29].

Large Language Model Meta AI (LLaMA): family of autoregressive large language models released by Meta AI starting in February 2023

NOTE: As defined in https://github.com/meta-llama/llama.

NetGPT: domain-specific model trained with data from core network domain

policy: set of rules that is used to manage and control the changing and/or maintaining of the state of one or more managed objects

NOTE: As defined in [i.29].

Quality of Experience (QoE): performance of users when using what is presented by a communication service or application user interface

Quality of Service (QoS): collective effect of service performances which determine the degree of satisfaction of a user of a service

NOTE: As defined in ETSI TR 121 905 [i.35].

Service Level Agreement (SLA): contract between a service provider and a customer that defines the service(s) to be provided and the level of performance to be expected

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

2B	To Business
2C	To Consumer
2H	To Home
ABI	Agent Based Interface
ADRF	Analytics Data Repository Function
AF	Application Function
AgentGPT	Agent Generative Pre-trained Transformer
AI	Artificial Intelligence
AI4N	AI for Network
AMF	Access and Mobility Management Function
API	Application Programming Interface
AR	Augmented Reality
BS	Base Station
CN	Core Network
CoT	Chain of Thought
CoT-SC	Self Consistency with Chain of Thought
DB	Data Base
E2E	End-to-End
eMBB	enhanced Mobile BroadBand
FB	Functional Block
GoT	Graph of Thought
GPT	Generative Pre-trained Transformer
HTTP	HyperText Transfer Protocol
ICT	Information and Communication Technology
IMU	Inertial Measurement Unit
IP	Internet Protocol
ISAC	Integrated Sensing And Communication
IT	Information Technology
KNN	K-Nearest Neighbour
KPI	Key Performance Indicator
KQI	Key Quality Indicator
LLM	Large Language Model
ML	Machine Learning
mMTC	massive Machine Type Communications
MSISDN	Mobile Subscriber Integrated Services Digital Network Number
N4AI	Network for AI
NetGPT	Network Generative Pre-trained Transformer
NF	Network Function
NFV	Network Function Virtualisation
NGMN	Next Generation Mobile Networks
NLP	Natural Language Processing
NRF	Network Repository Function
NWDAF	NetWork Data Analytics Function
O&M	Operation and Maintenance
OAM	Operations, Administration and Maintenance
PCF	Policy Control Function
QoE	Quality of Experience
QoS	Quality of Service
RAG	Retrieval Augmented Generation

RAN	Radio Access Network
RCS	Rich Communication Service
ReAct	Response and Action
RPC	Remote Procedure Call
SBA	Service Based Architecture
SLA	Service Level Agreement
SMF	Session Management Function
SMS	Short Message Service
TCP	Transmission Control Protocol
ToT	Tree of Thought
UE	User Equipment
URLLC	Ultra-Reliable Low-Latency Communication
XR	Extended Reality

4 AI Agents Based Next Generation Network Slicing

4.1 Revolution Trend

4.1.1 New Usage Scenarios of Future Mobile Network

The NGMN white paper [i.27] describes new use cases and services that need to be supported by 6G networks. These new use cases and services require 6G networks to provide ultra-high performance while connecting humans, machines and various other entities. This calls for a variety of new capabilities and requires 6G networks to have enhanced on-demand customization capabilities to adapt to the wide range of applications autonomously. These include immersive multimedia and multi-sensory interactions, highly intelligent industrial applications, integration of physical and virtual worlds through digital twins, and ubiquitous intelligence and computing.

As described by Recommendation ITU-R M.2160-0 [i.28], it is expected to integrate sensing and intelligence capabilities, empowered with AI and machine learning, into networks to keep up with the steady progress and fast spread of such. As stated in the same document, [i.28] could serve as an AI-enabling infrastructure that can provide services for intelligent applications listed above. Therefore, 6G networks are expected to face great challenges in the future as intelligent applications will take numerous forms and may be triggered based on user intents.

4.1.2 Potential Improvements of 5G Network Slicing

4.1.2.1 Network Slicing and NFV

5G systems are known for their heterogeneity in service categories, such as enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and massive Machine Type Communications (mMTC). Such a broad diversity in service requirements calls for customized solutions by 5G network operators for their customers, which has primarily been addressed via the network slicing concept. Together with Network Function Virtualisation (NFV), network slicing allows the 5G mobile network operators to build dedicated, virtualized and logical networks on a common physical infrastructure to meet the diverse communication requirements of their customers.

In contrast to 5G networks, which are designed for providing only communication service to their users, 6G networks are envisioned to extend their services beyond connectivity. More specifically, 6G networks are expected to add AI, compute, and sensing to their services on top of connectivity, introducing new types of resources, new functionalities, and design considerations. This calls for a more flexible, autonomous, and generalizable network slicing framework for the configuration, deployment and management of such slices of new type. Since this comes with increased complexity rendering the conventional methods insufficient, the deep integration of AI/ML technology into the operation of 6G networks offers a promising solution.

4.1.2.2 Network Data Analytics Function (NWDAF)

3GPP has introduced a new logical function entity in 5G, known as the Network Data Analytics Function (NWDAF), into the 5G network architecture [i.1]. The NWDAF can interact with other core network functions such as the Application Function (AF), Policy Control Function (PCF), Access and Mobility Management Function (AMF), and Session Management Function (SMF) to provide network data analytics services, network intelligence, and automation capabilities for 5G networks.

These services include receiving network data analytics requests from other core Network Functions (NFs), collecting and analysing data using AI algorithms to generate network analytics results, and delivering these results to the requesting (i.e. consumer) NF.

Each NF leverages the network analytics provided by the NWDAF to monitor the operational status of the 5G network and the User Equipment (UE), enabling closed-loop network control and optimization of communication services. For example, the NWDAF supports analytics and event exposure services related to network service experience, network performance, slice load, NF load, UE mobility, communication events, abnormal events, Quality of Service (QoS) sustainability, user data congestion, and more.

3GPP has enhanced the 5G network data analytics framework over multiple releases by introducing the logical functional division of the NWDAF and defining their interactions. It also facilitates cooperation between multiple NWDAF instances for model training and sharing, while incorporating new functional entities to improve data collection efficiency and enhance real-time performance [i.2]. Several enhancements have been proposed for the NWDAF, particularly to support flexible deployments (centralized, distributed, etc.), enable collaboration between different NWDAF instances, decompose NWDAF functionality, and achieve tighter integration with the UE (e.g. analysis of session load and signal quality). Additionally, the NWDAF has been further integrated with edge devices to optimize network operations.

Despite all of the aforementioned features, the existing capabilities of NWDAF primarily focus on analytics centred around better connectivity. As it has been explained in clause 4.1.2.1 on 6G network slicing, the considerations beyond connectivity service (e.g. compute, sensing, AI) are not fully incorporated into the NWDAF's services. For instance, idle resources within the network infrastructure will be important and needed to embrace the native integration of AI and deployment of foundation models towards an autonomous and optimized network operation. Therefore, 6G systems are to make better use of under-utilized resources in the network, contributing to sustainability targets and further expand the profitability of mobile networks for mobile network operators.

4.1.2.3 AI Agents

The current 5G core network deeply integrates telecommunications networks and IT technologies by cloud deployment, making the network architecture more agile and open. Therefore, network operations have become more efficient and automated. Looking towards the 6G era, in which the diversity of network services, resources and capabilities are envisioned to increase, standardized pre-defined processes based on scenario-specific expert knowledge are no longer sufficient for the efficient operation of the network. This is due to the significant increase in service and resource diversity required along with increased flexibility that is needed in service requirements and network functionalities. Together, these bring unprecedented challenges to the network architecture design, especially for the core networks.

Agentic AI is a new class of artificial intelligence systems designed to act with autonomy, making decisions and taking actions without having been specified in advance or without direct human intervention. These systems are capable of processing vast amounts of data, reasoning (the process of reaching understanding), and adapting to real-time changes in their environment. Hence, AI agents-based systems are a promising solution towards a more generalized and extensible design of 6G systems. Key features of Agentic AI systems include autonomy in decision-making, goal-oriented behaviour, continuous learning and adaptation, proactive planning and execution, as well as advanced reasoning capabilities According to [i.26], systems integrating AI agents "are characterized by the ability to take actions which consistently contribute towards achieving goals over an extended period of time, without their behaviour having been entirely specified in advance". This will render the mobile network a fully autonomous networks, as specified in [i.33], where any scenario (although unknown from before) can be supported due to excellent adaptability and knowledge reasoning capabilities of Agentic AI.

Developing and deploying an AI Agent is not a straightforward process, as such systems are complex by design requiring various functional modules and mechanisms to realize its value to the full extent. These typically include different decision-making models, vector databases, tool libraries, self-reflection and self-evolving mechanisms. As of today, AI agents have not been considered and deployed in 3GPP systems.

An example use case to be studied for the adoption of AI agents in 6G core is the optimal and flexible design of End-to-End (E2E) network slices. In such a setting, AI agents that are empowered with Large Language Models (LLMs) can be employed as an interface to support human operators in defining high-level intents and setting up optimization tasks as external input.

4.2 AI-Core Concept: AI Agents-Based Next Generation Core Network

4.2.1 AI Agent Introduction

AI Agent is defined as an autonomous system that can interact with its environment to collect data, learn from the past experiences and subsequently use these to improve its decision making capability in order to perform specific tasks [i.6], [i.7] and [i.8].

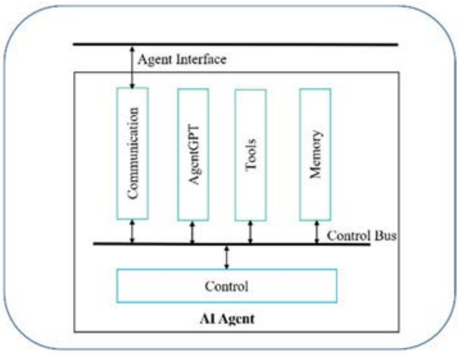

Figure 4.2.1-1: General Framework of an AI Agent

Figure 4.2.1-1 depicts a general framework of an AI Agent that is made up of the following logical components:

- Communication: The interface of an AI Agent to communicate with external components, supporting various networking standards and protocols, such as TCP/IP and HTTP. The communication block handles the input/output operations.
- Memory: Collects and stores data for the AI Agent for task continuity and self-improvement, including short-term memory (external input, historical inference result, temporary information, etc.) and long-term memory (knowledge, profile, etc.). The memory sub-component plays a crucial role in accelerating and agent's learning and adaptation capabilities, thereby, contributing to reducing computational complexity and energy efficiency. Such mechanisms have been used in the existing literature, not necessarily to design and implement AI agents, for storing/caching reoccurring problems that have already been solved in the past, also referred as dynamic programming [i.31], [i.34].
- AgentGPT: a domain-specific model with less parameters compared to the 'NetGPT' (see clause 4.4). The AgentGPT is trained with domain-specific knowledge (e.g. on a certain problem class) that matches the responsibilities of the AI Agent that it resides in.

NOTE 1: The name "AgentGPT" does not reflect any publicly available library or service and has solely been selected to emphasize the fact that it resides inside an AI Agent, also to easily distinguish from NetGPT.

NOTE 2: The name "AgentGPT" is also chosen because an AI Agent system does not mandate either the use of an LLM or an Agent to be embodied in an LLM. Since AgentGPT does both, the name has increased significance.

- Tools: Functions and APIs that are used to obtain additional information or abilities that are not present in the AgentGPT. These can include search engines, databases, calculators, calendars, maps, APIs for specific services, and other task-specific utilities.
- Control: The executive function of the agent that orchestrates the interaction between all components. It manages the flow of information, decides when to use AgentGPT or specific tools, coordinates memory access and updates, and implements the agent's overall strategy and decision-making.

Some open source frameworks (such as LangChain, LangGraph, CrewAI, Semantic Kernel, AutoGen) can be referred for the implementation of agent and multi-agent-based architectures [i.3], [i.8] and [i.24]. They can:

- Accelerate development by providing in built components;
- Provide consistent approaches to common challenges;
- Support scalability by moving from simple agent to complex multi agent environments;
- Access to a broad range of developers and researchers;
- Foster innovation by handling the foundational aspect of AI agent development frameworks.

All the frameworks above do have particular advantages and disadvantages. Significantly, none of them yet support specific use cases of telecommunication networks. Nevertheless, these frameworks can be seen as a baseline for agent-based core network developments.

4.2.2 Definition of AI-Core

The proposed AI-Core for the next generation core network consists of multiple agents. As shown in Figure 4.2.2-2, it utilizes multiple AI Agents to manage and control the network and to flexibly process the data for new services based on the dynamic requirements of various applications. Here, the multi-agent core assembles the network functions, application functions and resources autonomously to create the E2E slice. Moreover, it monitors its status in real time, dynamically updates the functions or resources of the slice when the network environment or demand changes, and deletes the slice instance when the service is terminated.

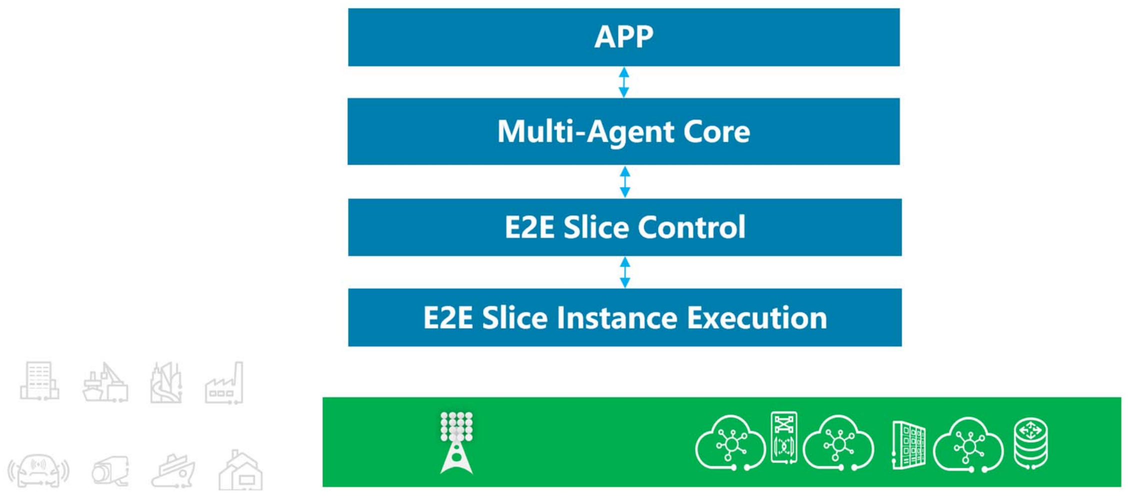

Figure 4.2.2-2: High-level conceptual illustration of the AI-Core

Different than the current 5G network slicing, the proposed network slicing using the AI-Core concept expands the scope of a network slice. Flexibly customizes network functions and application functions, as well as computing- and communication resources in an E2E fashion. This can provide abundant and diverse services for the slice tenant. In addition, the E2E customization greatly decreases the cost and complexity of slice customization for tenants. Moreover, AI-Core does not require customers (e.g. tenant/application/UE) to provide a large set of technical parameters that are used for the slice configuration. Instead, a high-level intent that is to be interpreted by the intelligent multi-agent component in the architecture above can be provided in natural language. This way, it is expected to have significant added value for the to consumer (2C)/to business (2B)/to home (2H) scenarios. The management and control by the multi-agent core for slice design and execution is highly intelligent and autonomous, which in return facilitates the cross-domain slice collaboration. Thus, the proposed AI-Core can also contribute to improving the network autonomicity, characterized by different levels as presented in [i.4].

It is important to mention that although the present document focuses on redefining the concept of E2E network slicing, the multi-agent-based AI core can be extended to various other use cases and decision-making problems in the network including but not limited to improved coverage, network capacity and energy efficiency, targeting the sustainability and ubiquitous connectivity considerations recommended in [i.28].

4.3 AI-Core Architecture and Interfaces

4.3.1 Design principles

The following design principles are used for the agent-based core architecture:

- Multiple AI Agents constitute the foundation of the core network architecture and are not employed as a simple "integration" or an "add-on" feature.
- The agent-based architecture serves as a reference for different network services (e.g. connectivity, compute for industrial applications or AI applications) with very limited to no modifications needed to tailor it for each individual service. Thus, it is inherently general and extensible.
- Agent reference architecture will support multiple technical and business domains.
- AI-Core complies with all relevant compliance, security and privacy obligations, including explainability as required by the AI Act [i.37].
- The resulting next generation core network offers significant benefits towards AI for Network (AI4N) and Network for AI (N4AI) services (internal and 3rd party).

4.3.2 AI-Core Reference Architecture

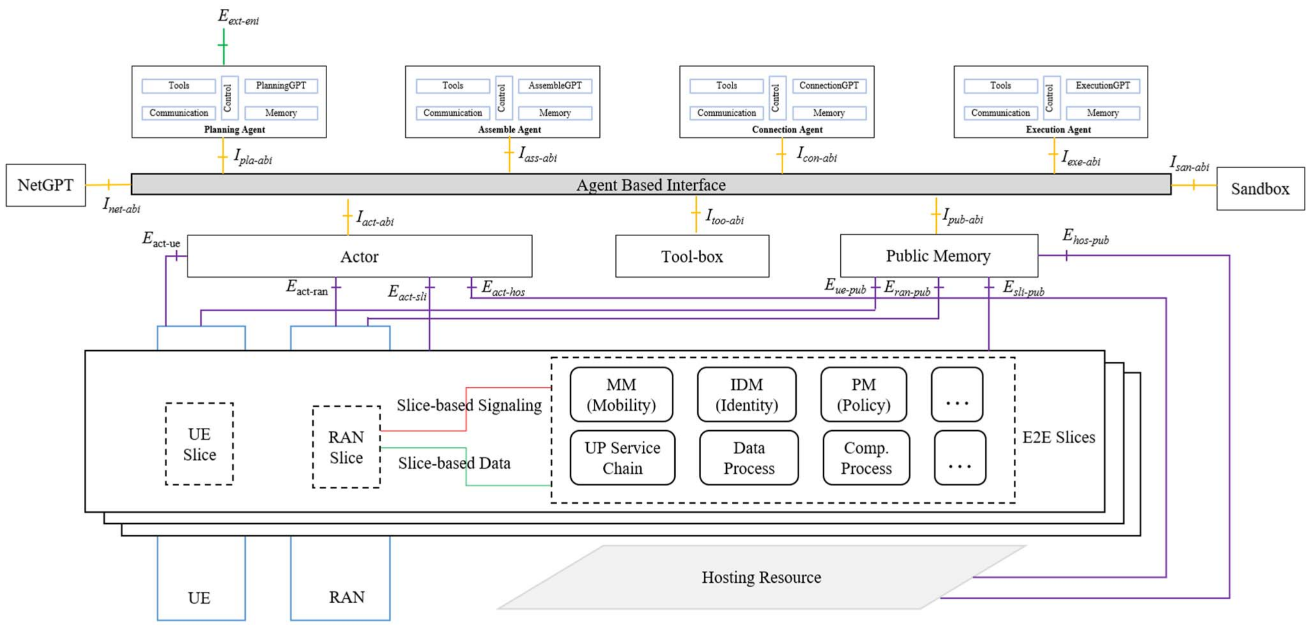

Figure 4.3.2-1: Reference Architecture of AI-Core

Figure 4.3.2-1 shows the AI-Core reference architecture. It consists of multiple agents that are optimised for different tasks, multiple common components that are shared by these Agents and an Agent Based Interface (ABI). The functionalities of Agents, common components and the ABI are explained in the following, while other reference interfaces are further described in clause 4.3.3.

- **Planning Agent** receives an input policy, information, and knowledge (e.g. a customized slice request from an application, which can be an imperative, declarative, or intent-based policy [i.29], [i.36]). It is the responsibility of the planning agent to convert or decompose the input (if necessary) into multiple executable policies or requests. Specifically, the planning agent outputs the decomposed task list, where each item includes a task description, task dependency indicators, QoS requirement, and more. This can be mapped to a combination of multiple functional blocks of an ENI system (e.g. a combination of the data ingestion and data normalization FBs, as specified in [i.29]).

- **Assemble Agent** is responsible for receiving the output (i.e. a decomposed task list) from the *planning agent*, identifying the appropriate network- and/or application functions for each sub-task, and subsequently determining the function topology based on the task dependency relationship. The assemble agent outputs two parts of function list, where the first part is a function list for configuring UE(s) and/or RAN(s), and the second part is intended for the deployment of other services beyond connectivity (e.g. computing services) that are necessary for the requested E2E slice. Each entry in the list includes a function ID, function input(s), a type (configure or deploy), and a function dependency indicator.

  NOTE 1: The output of the assemble agent does not specify the exact configuration of the access and core network configuration of the E2E slice components, which is the task of the connection and execution agents.

- **Connection Agent** is responsible for receiving the output part 1 (i.e. the function list) from Assemble Agent, determining which UE(s) and/or RAN(s) to be utilized in the requested E2E slice and determining their configuration parameters and resources, managing and controlling the connection topology of E2E slices, supporting the access management and slice selection for UEs. It outputs the instructions for the *actor* to configure UE(s) and/or RAN(s) and to connect the RAN with function instances.

- **Execution Agent** is responsible for receiving the output part 2 (i.e. deployment) from the *assemble agent*, managing and controlling the lifecycle of CN slices, including deploying the network and/or application function instances and chaining the function instances, monitoring the operational status of network slices, updating and recycling the network slices dynamically. It outputs instructions for the *actor* to deploy, update, and recycle function instances and chaining function instances. The key difference between the connection and execution agents is, while the connection agent configures, deploys and monitors the entities within an E2E slice that is related to providing the requested connectivity service, the execution agent handles the deployment of network and application functions that extend the scope of an E2E slice beyond connectivity (e.g. deployment of certain services on hosting resources as requested by the customer).

- **Actor** is responsible for translating the management and control instructions for E2E slices received from agents to the control signalling for UE(s), RAN(s), slices and hosting resources. It also translates the information from UE(s), RAN(s), slices and hosting resources to instructions and information for agents.

- **Toolbox** maintains the information or profile of available network functions and application functions (functions, models, agents, APIs, etc.) required for E2E slices, the function information or profile includes the function name, description, type, required parameters, optional parameters, output examples, usage method, etc. It can receive query or subscribe request for functions from other components and output the response or notify about the required function information or profile.

- **Public Memory** is responsible for collecting, vectorizing and storing the network data and knowledge that NetGPT does not have, as well as building the index for these data or knowledge. It can receive data query requests from other components, and output the requested data to the consumer. Thus, public memory supports other components (e.g. agents) by allowing them to retrieve data, information, and knowledge. Thus, public memory is an essential functional block within the multi-agent core for offering a common short- and long-term storage service for the entire multi-agent system. Moreover, it is important to mention that the public memory should support different communication paradigms, such as client-server, publish-subscribe, as well as different data modalities for data ingestion and retrieval, such as natural language, or binary.

- **NetGPT** is a domain-specific model, which can be based on an LLM, trained particularly with telecommunications and networking domain knowledge. It is responsible for assisting other AI agents for their operation and tasks. NetGPT is a larger model when compared to the AgentGPTs (within AI Agents). This means, while each AgentGPT is trained for one or a combination of sub-tasks within the overall agent-based core network, NetGPT is expected to have a higher inference accuracy for broader tasks, which are not necessarily within the domain knowledge of each AI agent.

  NOTE 2: Although different implementations are possible, AgentGPTs and NetGPTs are different models, meaning that they are not part of a single model. More details on the training and implementation of NetGPT can be found in clause 4.4.

- **Sandbox** is an isolated environment where code can be executed, tested, or run without affecting the rest of the system. The Sandbox is used to verify the feasibility and performance of network slices generated by AI agents.

- **Agent Based Interface (ABI)** is a functional block that is responsible for facilitating communication between various other entities. By leveraging advanced AI models and functionality (such as NLP, and intent-based approaches), the ABI supports imperative, declarative, and intent-based interactions. This enables it to understand the purpose of incoming messages and direct them to the appropriate agents or shared components. The ABI can also modify the message content before forwarding, for instance, in response to an incoming intent message via the defined reference points, i.e. Ixxx-abi.

  The ABI can operate as a Semantic Bus (see [i.29]). It also has proxy capabilities and can be implemented in various ways. Some of those can consider an internal model, either an LLM or a simpler, fit-for-purpose model, assisting this validation. It can also make use of past actions given similar requests cached in its internal memory (if it exists) or the *public memory* component of the AI core. Although the exact internal architecture of the ABI is left for future studies, it is important to note that it would be beneficial for the ABI to support also functionalities of a classical service bus, if requested.

In addition to the descriptions above provided for each component of the AI core architecture, clause 6 elaborates further on their role and showcases an example procedure for the E2E slice configuration. Clause 6 also includes an example flow diagram for better understanding of the interaction between agents and shared components.

Why is AI-Core a multi-agent system?

As shown in Figure 4.3.2-1, the AI-Core architecture contains multiple agents to manage and control E2E slices, instead of a single one. The reason behind the multi-agent architecture is that the E2E slice design and execution control are usually complex, and using a single agent leads to high implementation complexity and poor performance. For example, during the slice design process, the Agent needs to determine the working procedure and the required functions, the configuration parameters for functions and the required resources, according to the high-level service requirement description provided by customers. It also needs to assemble and chain the functions, compute the optimal data routing path. During the slice execution process, the Agent needs to monitor the execution status of the slice. When the network environment changes, it is required to adjust the functions or topology of the slice and schedule the resources dynamically. If a single agent is adopted to complete all the things, the functionalities of this Agent will be quite complex, which will lead to high implementation cost. Especially, the AgentGPT of the single agent will be powerful enough to solve a diverse set of problems, which implies that the number of parameters of the AgentGPT will be sufficiently large. The large-scale model will incur high training cost and large inference time. Besides, many academic researchers have demonstrated that the performance of a single agent is worse than the cooperation of multiple agents [i.9], [i.10], [i.11], [i.12], [i.13].

As a result, multiple agents are used, and each of them plays a different role and has customised small-scale models to solve specific problems. The collaboration of multiple agents and the introduction of closed-loop optimization mechanism among agents can improve the effectiveness and efficiency of slice management and control. As described and demonstrated in [i.30], a particular complex task can be successfully broken into simpler sub-tasks and solved through a multi-agent conversation, which improves the modularity and extensibility of the entire system. This also goes hand-in-hand with the "single responsibility principle" recommended in [i.29].

Components shared by multiple agents

Although each agent has its own functional components, (e.g. PlanningGPT has its own internal tools and memory to enable it to work intelligently and autonomously), the common components, including NetGPT, Toolbox, Public Memory, Actor and Sandbox, which can be shared by all the agents, are still needed. The Toolbox maintains all the network functions and application functions required for slices, and will be accessed by the Assemble Agent, Connection Agent, and Execution Agent when they select functions, configure functions of UEs and RANs, and deploy function instances, respectively. The tools in each agent are used to obtain extra information that the agent requires to complete its task. For example, if the input to the agent includes a location name, it can invoke the function in Tools to obtain the geographic coordination of the location. The Public Memory stores the data collected from the network, such as the capability and location of UEs and base stations, and the execution status of slices. These data can be shared by the agents and used to improve the performance of the agents. While the memory in each agent stores the context and temporary inference result of the AgentGPT during its working process, these data are private and used only by that Agent. NetGPT is a relatively large-scale model while the AgentGPT in each agent is a relatively smaller model in size. NetGPT can be used to assist the AgentGPT of each agent in inferencing. The output of an agent is usually the instruction in the form of natural language, which cannot be understood by traditional network entities. Thus, the Actor is required to translate the instructions from the agents to the signalling messages for the network entities (e.g. in RAN) or vice versa. Moreover, the accuracy of slices created by the agents is not deterministic (e.g. due to hallucinations). Thus, employing the Sandbox to verify their feasibility can further improve the performance of generated slices.

Agent Based Interface Design

For the design of the ABI, Table 4.3.2-1 overviews some of the most popular frameworks used in LLM-based multi-agent system development and the communication interfaces and protocols they employ. These constitute examples and a starting point for the design and implementation of the multi-agent core network architecture, depicted in Figure 4.3.2-1. One can see from Table 4.3.2-1 that the communication interfaces and protocols that are currently utilized for agent-to-agent communication, e.g. HTTP APIs, WebSocket, Remote Procedure Call (RPC), are compliant with the service-based interface, which is widely accepted and adopted by the 3GPP specifications, particularly for core network architecture [i.3].

Table 4.3.2-1: Popular frameworks for multi-agent system development

Framework	Description	Communication Interfaces
LangChain	A framework for building applications with LLMs that supports chaining prompts, memory, and interaction with external APIs.	API calls, Python function calls, message passing
AutoGen	A framework for automatic generation of agent behaviours and interactions, focusing on multi-agent conversation.	REST APIs, WebSocket, RPC
CrewAI	A platform for creating collaborative AI agents that work together to solve tasks and provide information.	WebSocket, HTTP REST APIs, message queues
MetaGPT	A framework that leverages multiple GPT-based agents to collaborate on tasks, enhancing interaction and adaptability.	HTTP REST APIs, WebSocket, gRPC
AutoGPT	An autonomous agent framework that allows agents to perform tasks with minimal human intervention, using LLMs.	HTTP REST APIs, WebSocket, gRPC
Langraph	A framework designed for building multi-agent applications, focusing on collaborative interactions among agents.	HTTP REST APIs, WebSocket, message queues

4.3.3 Reference Points

In this clause, the interfaces of the reference architecture shown in Figure 4.3.2-1 are described:

- **$I_{pla-abi}$**: Agent-based interface of the Planning Agent, which is used to invoke Planning Agent to do task planning.
- **$I_{ass-abi}$**: Agent-based interface of the Assemble Agent, which is used to invoke Assemble Agent to make function selection.
- **$I_{con-abi}$**: Agent-based interface of the Connection Agent, which is used to invoke Connection Agent to configure UEs and/or RANs, and build connections.
- **$I_{exe-abi}$**: Agent-based interface of the Execution Agent, which is used to invoke Execution Agent to deploy function instances on hosting resources and management the hosting resources.
- **$I_{net-abi}$**: Agent-based interface of the NetGPT, which is used to invoke NetGPT to do reasoning (the process of reaching understanding).
- **$I_{act-abi}$**: Agent-based interface of the Actor, which is used to invoke Actor to translate control signalling or instructions.
- **$I_{too-abi}$**: Agent-based interface of the Toolbox, which is used to query the information of network functions and application functions in Toolbox.
- **$I_{pub-abi}$**: Agent-based interface of the Public Memory, which is used to request Public Memory for storing data or obtaining data from Public Memory.
- **$I_{san-abi}$**: Agent-based interface of the Sandbox, which is used to request Sandbox to make verification.
- **E_{act-ue}**: the interface between Actor and UE that is used for configuring UE slices, including configuring functions and resources, etc.

- **E$_{\text{act-ran}}$**: the interface between Actor and RAN used for configuring RAN slices, including configuring functions and resources, etc.
- **E$_{\text{act-sli}}$**: the interface between Actor and slices used for lifecycle management of function instances in slices.
- **E$_{\text{act-hos}}$**: the interface between Actor and hosting resources used for scheduling network resources to run function instances.
- **E$_{\text{ue-pub}}$**: the interface between Public Memory and UE used for collecting the capability, location, and operational status of UE.
- **E$_{\text{ran-pub}}$**: the interface between Public Memory and RAN used for collecting the capability, location, and operational status of RAN.
- **E$_{\text{sli-pub}}$**: the interface between Public Memory and slices used for collecting the operational status of slice instances.
- **E$_{\text{hos-pub}}$**: the interface between Public Memory and hosting resources used for collecting real-time data, e.g. load conditions.
- **E$_{\text{ext-pla}}$**: the interface between Planning Agent and external entity used by the third party or UE application to request the AI-Core to create a customized network slice.

4.4 NetGPT: Collaborative Inference with AgentGPT

4.4.1 NetGPT Training

NetGPT is a domain-specific model, which can be based on LLM, designed specifically for the core network domain. In general, NetGPT can either be developed by training from scratch using mobile network data such as 3GPP specifications, signalling data, and Operations and Maintenance (O&M) data or by fine-tuning an existing LLM, such as Falcon and LLaMA with core network-specific data. Both approaches are valid and come with their own advantages and disadvantages. While a general-purpose LLM may not have been trained with domain-specific information on controlling a mobile network, it can offer advanced reasoning capabilities that can be leveraged. Both approaches are valid and come with their own pros and cons. While a general-purpose LLM may not have much pre-existing knowledge relevant to controlling mobile networks, it can offer advanced reasoning capabilities that can be leveraged.

The main differences between the NetGPT and the AgentGPT are: 1) NetGPT is a larger model when compared to an AgentGPT, 2) NetGPT covers a wider knowledge than the AgentGPT(s). This means that any AgentGPT can consult the NetGPT as necessary. NetGPT can help the AgentGPTs in their training and inference (i.e. collaborative inference).

NOTE: Similar to AgentGPT, the name NetGPT has also been selected to emphasise that this is a network-wide (i.e. shared) component of the AI Core. It does not correspond to any specific library or tool that is publicly available.

In the following sections, key details on how such models can be trained, what types of data can be utilized, expected model sizes, and the fine-tuning process are discussed.

1) **Data used for training:**

   Static data refers to information that remains unchanged within the system. Some prominent examples are 3GPP specifications, API definitions, historical data on user movement that have been recorded, static information on the locations of base stations, data centres and edge compute centres, and slice information. These data typically remain constant and serve as important resources for building knowledge. Having been trained with such data, NetGPT can understand the basic architectures of core networks, different Network Functions (NFs), APIs, the Service-Based Architecture (SBA) bus, how services are instantiated, locations of different data centres, and conventional methods that can be used to retrieve location information such as IP addresses and hostnames. As a result, the static data enables the NetGPT to gain the necessary capabilities for supporting an efficient configuration and operation of the multi-agent AI core.

Dynamic data, on the other hand, includes information that is subject to frequent changes. This includes but is not limited to current number of users, Quality of Service (QoS) analytics and network performance, load on hosting resources, cell-level KPIs, and UE movement and measurement reports. Integrating dynamic data is generally more challenging and requires a well-designed and functional Retrieval-Augmented Generation (RAG) mechanism. It is recommended to apply multiple mechanisms to augment the capabilities of the LLM. For example, if a well-trained and finetuned LLM is the base layer, then RAG could be a middle layer to supply current information without changing the model, and the top layer could consist of in-context learning to provide immediate adaptation using a given context. The extent to which the dynamic data can be incorporated into training, in-context learning (using prompts), or via RAG for real-time knowledge updates is still an open challenge. The difficulty in integrating dynamic data leads to varying NetGPT slice lifecycle control lengths, which can range from minutes to days depending on the specific use case and service requirements.

2) **Data collection and cleaning:**

One of the advantages of using a transformer-based model like NetGPT is its ability to identify various dependencies and relationships between data from different network entities, even if the data is not processed (which qualifies it as "information" according to [i.32]). For NetGPT to gain a deeper understanding of the network, it may need access to raw data, for instance, retrieved using Operations, Administration, and Maintenance (OAM) systems. However, data cleaning and filtering are still very important for improving learning efficiency and inference accuracy. Typically, this process comprises the following steps:

- **Remove user-specific and encrypted data**: In many sectors, the available data are sensitive and will follow strict regulations. Therefore, before the training or fine-tuning process can begin, appropriate information privacy concerns are strictly followed to avoid sensitive information being included in the training process.
- **Remove dynamic information**: For example, IP addresses, especially private ones from internal networks.
- **Enhance data with metadata**: Add different types of underlying information. For example, the model needs to distinguish between a request header, request body, and request results. Providing the raw data without this differentiation may confuse the model during training. Including metadata alongside the data will help the NetGPT better understand the roles of different network nodes elements in request/response interactions.

3) **Fine-tuning:**

Fine-tuning a pre-trained model improves the inference accuracy for specific tasks. In addition, it can facilitate the adaptation of a model to dynamic settings. For the fine-tuning process, it is crucial to establish well-defined reward functions. These can be based on QoS, service experience, task completion rate, system failure rate, and so on. It is also important to identify possible new metrics for the multi-agent AI core to evaluate the performance of the NetGPT at assisting other agents and entities in their operation to capture the usefulness of the instructions generated by the NetGPT. Various methods can be used for the fine-tuning including supervised fine-tuning based on task-specific labelled dataset, reinforcement learning based on the collected network data as well as human feedback.

4.4.2 Collaborative Inference between AgentGPT and NetGPT

Although the AgentGPT (PlanningGPT, AssembleGPT, ConnectionGPT, ExecutionGPT) in each agent is good for solving problems of specific scenarios with local knowledge, it maybe not be able to reach the desired confidence level under some circumstances. This could be resolved by the collaborative inference with the larger model NetGPT through an efficient algorithm.

Speculative Decoding is a widely used technique to speed up inference for LLMs without sacrificing quality [i.19], [i.20], [i.21] and [i.22]. When performing inference, a smaller draft model is used to generate speculative tokens. The target LLM then verifies the output of the draft model and only outputs tokens that match its output. By leveraging faster inference of smaller draft models, speculative decoding turns autoregressive decoding on the target LLM into a more hardware-friendly batched operation, thereby increasing throughput while preserving accuracy.

Speculative decoding can be adopted to enable collaborative inference between AgentGPT and NetGPT, where AgentGPT acts as a smaller draft model while NetGPT act as a verifier for the AgentGPT.

4.5 Toolbox: Network and Application Function Collection

Toolbox is an extensible collection of network functions, application functions (including functions, models, APIs) and information. Developers can register their developed functions and models to Toolbox for use by agent. It greatly enriches the variety of customized slices. The functions in Toolbox can be non-standardized. Toolbox can be an independent new function, an enhancement to the 5G Network Repository Function (NRF), or it can be open source.

4.6 Actor: Communication Bridge between Agents and Traditional Network Entities

As mentioned above, Agents are responsible for generating intelligent decisions and instructions related to network slices. The Actor serves as the communication bridge between agents and traditional network entities. Agents may use natural language while traditional network entities use symbolic language (e.g. binary or special purpose text as used in command line interface systems). The Actor translates the instructions of agents to the standard control signalling that can be understood by network entities, including UEs, RANs, traditional network functions, some common components (such as Toolbox, Public Memory) and underlying hosting devices. In addition, the Actor can translate the signalling from network entities and hosts to instructions that can be understood by agents.

4.7 AI-Core Data Management via Public Memory

As mentioned earlier, the Public Memory collects, vectorizes and stores various network data and knowledge. It has long-term memory and short-term memory. The data sources include the UEs, RANs, E2E slice instances, and hosting resources. The capability, location, status, and availability information from UEs and RANs are collected through $E_{ue\text{-}pub}$ and $E_{ran\text{-}pub}$ interfaces, respectively. They can be stored in the long-term memory and used for future queries by the agents. The operational status and the function output results of E2E slice instances are collected via the $E_{sli\text{-}pub}$ interface. The utilization of hosting resource is collected through the $E_{hos\text{-}pub}$ interface. These data are stored in the short-term memory, and they can be accessed and used during the execution process of the E2E slices. Furthermore, Public Memory can vectorize the collected data and build a vector database to store the vector data to facilitate fast data retrieval and efficient similarity search.

The collected data of Public Memory can be used to improve the capability of NetGPT and AgentGPTs. There are multiple ways to make use of the short- and long-term memory service offered by the Public Memory. An example is to provide assistance information for NetGPT and AgentGPTs upon request for Retrieval Augmented Generation (RAG) functionality. Another alternative is training scenario-specific reward models based on the collected data and update the generation policy of NetGPT and AgentGPT in each agent, e.g. through reinforcement learning.

4.8 Slice Performance Improvement Mechanism

4.8.1 General Slice Performance Improvement Mechanism

Several ways are recommended in the following clauses to improve the performance of generated slices, including prompting, multi-layer closed-loop optimization mechanism, verification through sandbox, and human-in-the-loop.

4.8.2 Prompt Design

In order to let NetGPT or AgentGPT (e.g. PlanningGPT, AssembleGPT.) generate a more accurate inference result, prompt engineering is usually needed to guide the model to solve complex reasoning problems. Currently, the main prompting techniques include Chain-of-Thought (CoT) [i.14], Self Consistency with CoT (CoT-SC) [i.15], Tree of Thought (ToT) [i.16], Graph of Thought (GoT) [i.17], and Response & Act (ReAct) [i.18], etc. They are suitable for different wireless scenarios:

a) CoT: This is useful for solving complex problems that require a step-by-step reasoning process. It can be used for solving problems such as optimizing network configurations or predicting network behaviour. CoT helps in breaking down big problems into smaller and manageable steps to reach a final solution.

b) CoT-SC: This enhances the robustness of CoT by generating multiple reasoning paths and selecting the most consistent answer. It is particularly helpful in situations where the network data may lead to ambiguous or uncertain conclusions such as in anomaly detection. Examples include multi-step arithmetic reasoning in network planning, complex resource allocation across multiple network slices, predictive maintenance based on multiple factors, traffic prediction and load balancing, and QoS optimization considering multiple parameters.

c) ToT: This extends CoT into a tree structure, allowing for exploring multiple reasoning paths. In addition to being able to solve more complex examples than CoT and CoT-SC would struggle with, ToT could be used to explore multiple design options simultaneously, evaluating different network topologies and configurations in network planning and optimisation, providing multi-objective QoS optimisation where the objectives conflict with each other, or finding optimal routes for data packets in a dynamic environment.

d) GoT: This is built on ToT by considering interconnected reasoning paths, which may share nodes or interdependencies. GoT can be helpful in determining interdependencies between network components. Exemplary tasks include complex network troubleshooting, cross-layer optimisation, network security analysis, traffic engineering, and load balancing.

e) ReAct: ReAct agents combine reasoning and action in a continuous loop, are particularly well-suited for certain wireless network tasks that require dynamic decision-making and interaction with the environment. Examples include real-time network monitoring and troubleshooting, adaptive resource allocation, and dynamic network slicing management in highly changing environments. They are best suited for automated systems that need to react quickly with respect to changes in network conditions.

For more detailed information of these techniques, one can refer to [i.5].

The few-shot prompting technique is particularly useful in scenarios where extensive training data is unavailable and/or a specific output structure or format is required. Considering the diverse requirements of various slice services, one can design different examples for different service requirements. Few-shot prompting provides examples to guide the model's performance, which can be integrated into the more complex reasoning frameworks described above in this clause. A few-shot example library is maintained, which contains various examples corresponding to different types of service requirements. These examples can be handwritten or generated by models. Once the agent receives a request, it first uses the clustering algorithm such as K-Nearest Neighbor (KNN) to match multiple examples that are similar to the request from the few-shot example library, and then includes these examples in the prompt template to invoke the AgentGPT or NetGPT.

4.8.3 Multi-layer Closed-loop Optimization

AI-Core is a multi-agent system where multiple agents cooperate to generate E2E slices.

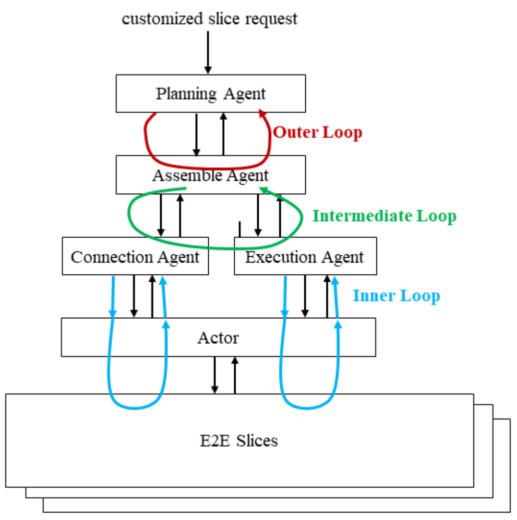

Figure 4.8.3-1

The three-layer closed-loop optimization mechanism among agents is utilized to enhance their performances, which are as follows:

- **Outer-Loop**: After the Assemble Agent receives the decomposed task list from the Planning Agent, it will select proper application and network functions to execute these tasks. If no function can be found for a certain task, the Assemble Agent will ask the Planning Agent to re-plan the tasks. The communication between the Planning Agent and Assemble Agent continues until all the decomposed tasks are assigned an appropriate function(s) or the number of communication rounds reaches the maximum preset value. If the number of communication rounds expires, the Planning Agent responds to the tenant that the network cannot meet this requirement.

- **Intermediate-Loop**: After the Connection Agent and Execution Agent receive the configuration or deployment function list from the Assemble Agent, the Connection Agent will decide the proper UE(s) and RAN(s) to configure the function and the Execution Agent will deploy the function instances on the proper hosts. If the Connection Agent or Execution Agent find insufficient resource to configure or deploy the function(s), it will request the Assemble Agent to re-select function(s) that consumes fewer resources. The communication between the Assemble Agent and Connection Agent or Execution Agent continues until all the functions are successfully configured or deployed or the number of communication rounds exceeds the maximum value. If the number of communication rounds expires, the Connection Agent responds to the tenant that the network cannot meet this requirement.

- **Inner-Loop**: During the running process of the slices, if the network environment is changed (e.g. the UE is powered off or the BS becomes busy or the hosts are highly loaded), the Connection Agent will autonomously replace the UE or update the slice topology and the Execution Agent will dynamically reschedule the hosting resources to adapt to the change of network environment.

4.8.4 Sandbox

In agent-based systems, the inference accuracy and the success rate in task completion heavily depend on the training process and the ability of agents to understand the task and to break it down into a set of correct sub-tasks. An additional method to improve the inference accuracy is sandboxing, which allows the agents to evaluate the inference accuracy in an isolated and controlled environment [i.23], [i.24] and [i.25].

In our reference architecture, the sandbox component has the ability to run a system-level evaluation first to check the feasibility and validity of the input request and agent output coming from the ABI. For the specific use case of E2E slice management, this is enabled via:

i) the access of the sandbox to up-to-date information about the underlying network, such as network functions, existing slice configurations and active network nodes;

ii) background knowledge about slicing functionality which includes but is not limited to the composition of necessary sub-tasks in E2E slice lifecycle management, the network nodes and functions that are involved.

Subsequently, the sandbox component evaluates the performance of the incoming request (e.g. comprised of sub-tasks) through scenario emulation in order to assess its performance, which quantifies the inference accuracy prior to deployment. This may occur at the sub-task level or at the intent-level depending on the exact request. If the evaluation result is found to be inadequate (e.g. infeasible, invalid, not satisfying the requirement) the result is returned to the requesting agent via the agent-based interface. This way, the sandbox component serves as a valuable tool for preventing the propagation of inference errors from one agent to another, thus facilitating the closed-loop network optimization and automation.

4.8.5 Human-in-the-loop

Although the goal of using AI agent-based systems for network slicing is to fully automate the slice lifecycle management, human involvement remains crucial, particularly during the design and initial deployment phases. The AI agent requires time to learn and adapt to the specific requirements, and during this period, human operators assist in guiding the AI to make informed decisions. Human operators will also be required in following stages, since networks have dynamic requirements and an infinite number of possible configurations that have complex interdependencies. In addition, there are unforeseen scenarios, and the network is subject to unforeseen business policy changes. These and other factors demand continuing human expertise.

"Human-in-the-loop" refers to the close integration of human expertise into a task, such as slice lifecycle control and management. Human oversight is critical to ensuring successful end-to-end service delivery. In practice, human in the loop support can assist at several stages:

- During the design phase: Human operators define the key requirements and templates that the slice will fulfil. For example, human operators establish the Key Performance Indicators (KPIs), Key Quality Indicators (KQIs), and Service Level Agreements (SLAs) that the provisioned slice will meet. This enables the AI agent to better understand and address the necessary aspects of the design.
- During provisioning: Once the AI agent provisions the slice, it can validate the slice against these SLAs, KQIs, and KPIs, with the operator providing rapid feedback on the slice's performance.
- During runtime for anomaly detection: The AI agent continuously monitors the slice's runtime characteristics. If anomalies are detected, the AI alerts the human operator, who can then make decisions based on their expertise, even guiding the AI on the quality of its decisions.
- During slice decommissioning: The AI agent manages the decommissioning process and provides key performance metrics to the human operator, which can help improve the overall system design and performance.

To enable effective human interaction within the end-to-end slicing loop, specific APIs and interaction points will be defined. This is essential for setting clear boundaries between tasks assigned to human operators and those delegated to AI agents. These APIs can help the industry develop interoperable systems that enable integration between AI agents and human operators, as well as more effective distribution of responsibilities. Key interaction points between human operators and AI agents include:

- Input to the AI agent: human operators interact with AI agents through management consoles or APIs to define network intents, SLAs, policies, and security parameters.
- AI Processing: based on the inputs and predefined policies, AI agents autonomously manage network optimization, fault detection, resource allocation, and slice orchestration.
- Monitoring: AI agents provide continuous feedback to operators through real-time dashboards and monitoring APIs, ensuring transparency in decision-making.
- Human Intervention: when needed, operators can intervene via APIs to override decisions, adjust parameters, or offer feedback for AI model refinement.

5 Application Scenarios

5.1 Introduction to Application Scenarios

This clause gives a brief overview of some exemplary use cases and applications that are envisioned to undergo significant improvements through advances in machine learning and Agentic AI systems research. A detailed analysis and study on the selected use cases (and others) is left for future studies and work items.

5.2 Robots

The utilization of AI combined with agent-based network services is opening new frontiers in robotics. As both the network and robots become more intelligent and autonomous, new example use cases in the field of cloud robotics are envisioned to appear, transforming various vertical sectors from industrial automation, healthcare, agriculture and mining, in which networked robots perform various tasks in a more intelligent, collaborative and responsive manner. With an unmatched capability of real-time problem solving, AI-empowered robots will tackle complex and novel problems in dynamic environments, such as adaptive motion planning, task offloading, collaborative task execution (i.e. collaborative robots). This will not only call for improved connectivity services, but an increased demand for E2E computing, sensing and storage services offered by the proposed AI-agent-based core network.

5.3 Sensing

Integrated Sensing and Communication (ISAC) is a key scenario of next-generation networks that enables new applications and services by leveraging network sensing capabilities. Typical use cases include network-assisted navigation, activity detection (e.g. vehicle and pedestrian), movement tracking (e.g. posture and gesture recognition, fall detection), environmental monitoring (e.g. rain and pollution detection), and providing sensing data for applications such as AI, extended Reality (XR), and digital twin technologies.

In addition to the need for advanced communication capabilities, ISAC requires support for high-precision positioning and sensing. A fast compute layer that supports rapid range/velocity/angle estimation, object and presence detection, localization, imaging, and mapping is essential. Furthermore, each use case has different KPI requirements, such as velocity/position estimation accuracy, reconstruction/imaging accuracy, missed detection probability/false alarm probability, and resolution (distance, angle, velocity). Such a complex KPI-to-feature mapping requires an advanced core, such as the AI-Core presented in the present document.

Beyond KPIs, the actual data collection procedures and exposure functionalities depend on the scenario. To address this, the AI-Core is capable of providing a flexible sensing service to adapt to new and unforeseen usage scenarios with varying requirements. Leveraging an AI-core network can provide the necessary flexibility, ensuring the network can adjust to evolving demands with a wide range of sensing capabilities.

5.4 Smart City

Due to the rapid growth of population and the increase in the number of vehicles on the road, most cities are faced with the problem of traffic congestion. AI-Core can be used to effectively help build and manage sustainable transportation systems, such as real-time access to vehicle information, intelligent decision making for planning, scheduling, and management.

An example use case that takes advantage of the AI-Core is the following. During peak commuting hours, smart city operators need to know the real-time information of the vehicle flow in various spots (e.g. scenic and other points of interest) to conduct intelligent traffic dispatch, alleviate traffic congestion, and improve the travel convenience of mobile users. In this case, the smart city operators may require the mobile network to provide customized network slices to monitor the vehicle flow during the peak time. After the peak commuting hours, the monitoring service can be terminated automatically, required to be terminated by the operators, or even repurposed. Further, the dedicated network slice for monitoring service is deleted or repurposed accordingly. Thus, the creating and reclaiming of these customized network slices for services are on-demand. AI-Core is a good way to provide customized network slice since the agent has strong capabilities of intention understanding, thought generation and tool usage.

5.5 Smart Network

With the continuous advancement of communication technology, wireless core networks are becoming increasingly large and complex, encompassing a vast array of devices, protocols, and service domains. This complexity renders traditional manual operations inefficient and prone to errors. Network faults often occur suddenly, and traditional manual response methods struggle to react swiftly. Intelligent agents, through real-time monitoring and immediate analysis, can rapidly detect and address faults, minimizing business downtime. In addition, intelligent agents can be trained on historical fault tickets based on business logic, identify and classify faults, and through intent recognition, search the knowledge base for maintenance manuals, safety regulations, etc., ultimately recommending fault resolutions and safety tips. This empowers frontline production maintenance, providing solution references for fault handling by frontline personnel, thereby enhancing maintenance efficiency and quality.

1) Automated slice design: Intelligent agents can automatically complete the design and configuration of network slices based on business requirements and network conditions. This includes resource allocation, QoS settings, and SLA guarantees.

2) Real-time monitoring and dynamic adjustment: Intelligent agents possess perception and prediction capabilities, allowing them to monitor the status of network slices in real-time and dynamically adjust slice configurations based on actual traffic and user demands, ensuring continuous service optimization.

3) Scene self-learning ability: Through scene self-learning, intelligent agents can continuously optimize network slice management strategies, improving network resource utilization and service flexibility.

5.6 B2B Voice and Data Services

With the improvement of LLM and Agent technology, it is possible to use personal assistants to improve the call experience. With these assistants, the following can be imagined:

1) When users A and B are in a call, user A interacts with the assistant using a natural language interface, and the assistant helps in complete tasks according to the user's call intention, for example, User A's schedule and location, and then booking a dinner reservation for users A and B.

2) Users A and B discuss the sales plan for about half an hour. During the discussion, three outstanding action points are generated and need to be tracked and handled later. The assistant is activated before the end of the call, to create summary minutes, and sends the minutes to A and B in Short Message Service (SMS) or Rich Communication Service (RCS) messages.

3) When user A is on the plane and cannot answer a call, the personal assistant of user A senses the status of user A, actively answers the call of express delivery, and informs the courier to put the express delivery on the small round table at the door according to the preferences of user A.

4) Call assistant identifies that the calling number is a real estate sale call. It answers the call, attempts to understand the call intention, forms a summary, and sends SMS or RCS message to notify the subscriber.

In addition to above call scenarios, a personal agent can be also used improve user experiences with new technology such as immersive Augmented Reality (AR) glasses with intelligent capabilities. When a user wears lightweight AR glasses with a Mobile Subscriber Integrated Services Digital Network Number (MSISDN), information such as images and videos is transmitted to the personal agent on the network side along with voice, and the multi-modal information understanding can help the user experience brought by the new technology. The scenario is as follows: A blank subscriber has a conversation based on AR glasses and asks an assistant to assist in arriving at the station from home. In this process, the network personal assistant identifies the surrounding environment, observes whether there are passing cars, reminds the user to cross the road in time, determines the direction of progress according to the visual information and Inertial Measurement Unit (IMU) information, and feeds back the route and obstacles to the user in real time.

6 AI-Core E2E Procedure

6.1 E2E Slicing Lifecycle

The tenant sends the customized slice request to the Planning Agent which is required to decompose the request into a set of simple and executable tasks. After receiving the request, the Planning Agent-Control first invokes PlanningGPT to understand and translate the intent of the tenant, then it queries the Memory in the agent for any historical task planning record corresponding to the request. If there is a historical record in Memory, it can be used directly; otherwise, the Planning Agent-Control invokes the PlanningGPT to output the task list about how to satisfy the request or it generates a prompt and sends it to the NetGPT to assist the PlanningGPT in reasoning to understand and implement the task list. Each task information includes the task description, QoS requirement, indicators of dependency on other tasks, and may include other details. Finally, the Planning Agent outputs the decomposed task list. The decomposed tasks can be checked by human experts before they are sent to Assemble Agent.

The Assemble Agent is required to select and assemble proper network functions and application functions to form an E2E slice according to the received task list. To achieve this goal, Assemble Agent-Control first queries the local Memory for the historical function record for each task. If there exists a function record for a certain task, it does not need to reselect a set of functions. For the tasks that do not have a matching function record in the Memory, the Assemble Agent-Control obtains the information about the available network and application functions from the Toolbox. Then the Assemble Agent-Control invokes the AssembleGPT to infer the proper set of functions for each task or it requests NetGPT to collaboratively inference with AssembleGPT. The output result of the Assemble Agent contains two parts. The first part is the function list for configuring UEs and/or RANs, and the second part is the function list required for deployment. Each function information of the list includes the function ID, type, function input(s), function dependency indicators, and other required information. The Assemble Agent sends the first and the second part of output to the Connection Agent and Execution Agent respectively, after they are verified to be reasonable by Sandbox.

After receiving the function list for configuration, the Connection Agent will determine which UEs and/or RANs to configure those functions, and decide the required parameters (e.g. input parameters, algorithm configuration parameters, session-related parameters) and resources for them. Connection Agent-Control can query the detailed description, required parameters, optional parameters, and output example from Toolbox according to the received function IDs. Then, it obtains the capability, location, status, availability information of UEs and RANs from Public Memory, and it invokes the ConnectionGPT to infer the proper devices and configuration parameters. Then, the Connection Agent outputs the configuration instruction to the Actor. The Actor translates the configuration instruction to the control signalling for UEs and RANs.

The Execution Agent is responsible for instantiating the functions to be deployed on the hosts, chaining these function instances, and configuring the required parameters and the resources. Similarly, the Execution Agent-Control could query the detailed description, required parameters, optional parameters, function code, and output example from the Toolbox according to the received function IDs. It obtains the status (e.g. load, distribution, etc.) of each hosting resource from Public Memory, then it uses ExecutionGPT to determine the parameters, resources and routing path of functions. Finally, the Execution Agent outputs the deployment instruction to the Actor. The Actor translates the deployment instruction to the control signalling for hosts.

During the execution process of the slice, the UEs, RANs, and hosting resources report the execution data and status to the Public Memory. The Connection Agent and Execution Agent can continuously monitor the operational status of the slice, the network environment, and the status of hosting resource by querying the Public Memory. If the network environment or the status of the hosting resource is changed, the Connection Agent and the Execution Agent will adjust the connection topology and the hosting resources dynamically.

After the service provided by the slice terminates, the slice instances need to be deleted to release the resources (or alternatively, repurposed to serve different users).

6.2 E2E Flow Diagram

The inputs and outputs of each agent and common component are listed in the following Table 6.2-1.

Table 6.2-1: Overview of Agents and Common Components

Agent & Common Component	Functions / Goals	Input	Output	Impact
Planning Agent	Understand the customer's requirement for customized slice, and decompose the requirement into multiple executable tasks	Customized slice request, which is in the form of a high-level intent policy (e.g. provide a customized slice for monitoring the number of vehicles and people at Shanghai Century Park No.1 gate from 8:00 to 17:00.)	Decomposed task list, the information of each task includes task description, task dependency indicators, QoS requirement, etc.	New NF
Assemble Agent	Select proper network entities and/or application function(s) for each task and determine the function topology based on task dependency relationships	Output from Planning Agent	Part 1: Function list for configuring UEs and/or RANs, each including function ID, input parameters, type (configure), function dependency indicator, etc. Part 2: Function list for deployment, each including function ID, input parameters, type (deploy), function dependency indicator, etc.	New NF

Agent & Common Component	Functions / Goals	Input	Output	Impact
Connection Agent	Determine which UE(s) and/or RAN(s) to use the E2E slice and their configuration parameters and resources, manage and control the connection topology of E2E slices, support the access management and slice selection for UEs, etc.	Output Part 1 from Assemble Agent	Instruction for Actor to configure UE(s) and/or RAN(s) and build connection between RAN and deployed functions	New NF
Execution Agent	Lifecycle management of CN slices, including deploying the network and/or application functions and chaining functions, monitoring the operational status of network slices, updating and recycling the slice instances dynamically	Output Part 2 from Assemble Agent	Instruction for Actor to deploy/update/recycle functions and chaining those functions	New NF
Actor	Translate the instructions from agents to the control signalling to UEs, RANs, slices and hosting resources; and translate the signalling from UEs, RANs, slices and hosting resources to instruction for agents	Instructions from agents for creating, deploying, updating, or recycling a slice instance; or control signalling from UEs, RANs, slices, and hosting resources	Control signalling for UE(s), RAN(s), slices, hosting resource; or instruction for Agents	New NF, 3GPP interfaces between Actor and UEs/RANs/slices, open source interface between Actor and hosting resources
Toolbox	Maintain the information or profile of available network functions and application functions (functions, models, agents, APIs, etc.) required for E2E slices, including function name, description, type, required parameters, optional parameters, output example, usage method (e.g. http), etc.	Query or subscribe request for network and application functions	Responses or notifies about the required function information or profile	New NF or enhanced NRF or open source
FPublic Memory	Collect, vectorize and store the data or knowledge from network, build the index for these data or knowledge	Data storage request or data query request	Storage or query response about the required data	New NF or enhanced NWDAF
NetGPT	Generate the reasoning result for slices according to the input prompt, including task planning or function matching result, etc.	Prompt for generating required result, including goal description, input information, examples of output	Reasoning result	Open source publication
Sandbox	Verify the feasibility and performance of network slices generated by Agents	Information of generated E2E slices, including functions, resources, inputs, slice topology	Potential results of network slices and a large amount of data related to the potential results	Open source

As shown in Figure 6.2-1, after the Planning Agent receives the customized service request, the overall E2E procedure can be divided into three main phases: the slice design phase, the slice deployment and execution phase, and the slice recycling phase.

NOTE 1: The procedure below and the corresponding details in the subsequent figures are only for explanation and presentation purposes. They are intended to provide a better understanding of the AI-Core architecture presented in clause 4.

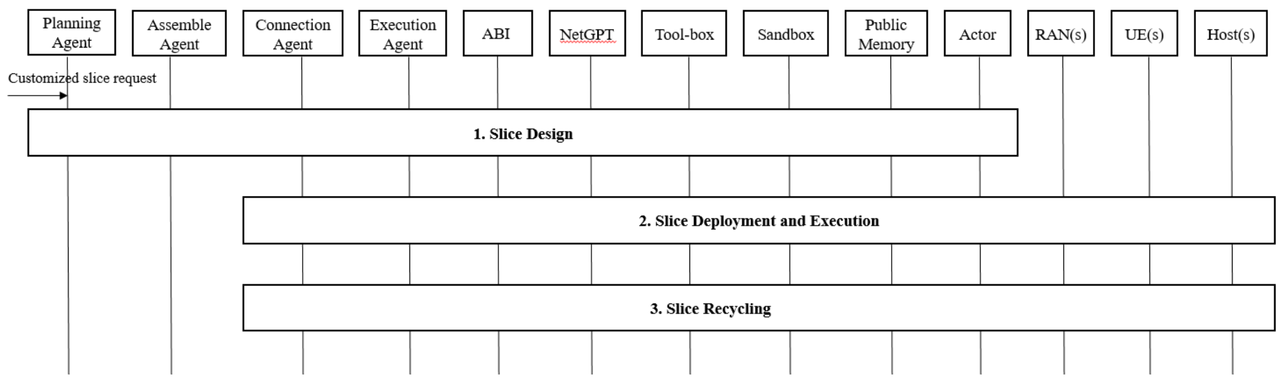

Figure 6.2-1: An overall E2E procedure

Figure 6.2-2 depicts the slice design procedure, the key points of which can be summarized as follows:

1) Upon the reception of a custom E2E slice creation request, the planning agent can perform a collaborative inference with the help of the NetGPT.

NOTE 2: This is an optional step, thus, indicated by a dashed arrow.

2) The Planning Agent decomposes the slice creation request into multiple tasks and forwards these to the Assemble Agent.

NOTE 3: The messages go through the ABI operating as a semantic bus as explained in clause 4.

3) Assemble Agent requests the list of available functions with descriptions, and parameter information from the Toolbox shared component.

4) The Assemble Agent composes the (candidate) functions of the requested E2E slice based on the received functions and descriptions. The selected functions are verified through sandboxing and accepted after validation.

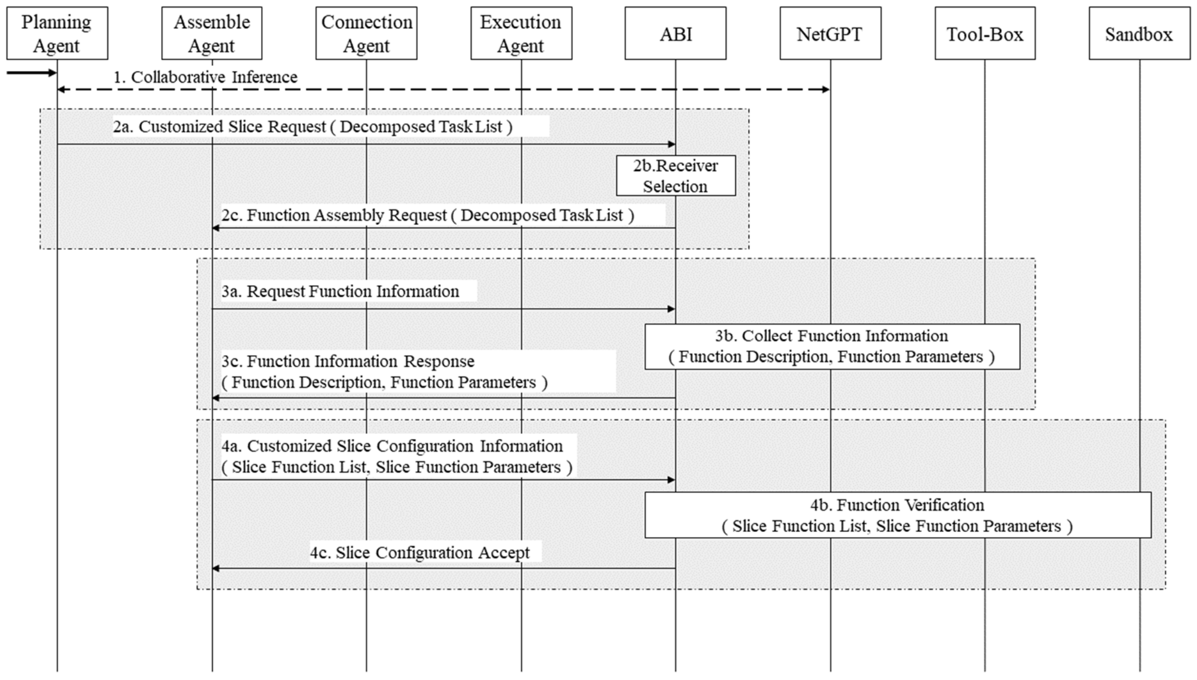
Planning Agent
Assemble Agent
Connection Agent
Execution Agent
ABI
NetGPT
Tool-Box
Sandbox
1. Collaborative Inference
2a. Customized Slice Request (Decomposed Task List)
2b.Receiver Selection
2c. Function Assembly Request (Decomposed Task List)
3a. Request Function Information
3b. Collect Function Information (Function Description, Function Parameters)
3c. Function Information Response (Function Description, Function Parameters)
4a. Customized Slice Configuration Information (Slice Function List, Slice Function Parameters)
4b. Function Verification (Slice Function List, Slice Function Parameters)
4c. Slice Configuration Accept

Figure 6.2-2: Slice design procedure

The procedure describing the deployment of slice execution functions is depicted in Figure 6.2-3:

5) After the validation of the determined slice functions and parameters (in step 4 as above), the ABI extracts the execution functions from the slice function list and provides it to the Execution Agent. The Execution Agent requests relevant information about the hosting resources, which have already been made available to the Public Memory. This happens in parallel in regular intervals and is not explicitly illustrated in Figure 6.2-3 for presentation purposes.

6) The deployment of the slice execution functions on the hosting resources is initiated in the sixth step, through message 6a. Here, one can see the role of the Actor clearly, which is translating the commands received from the agents (e.g. Execution Agent) to lower-level signalling messages (based on pre-determined APIs) that can be understood by the receiving entity. Once the sixth step is completed, the execution functions are already up-and-running on the hosting resources.

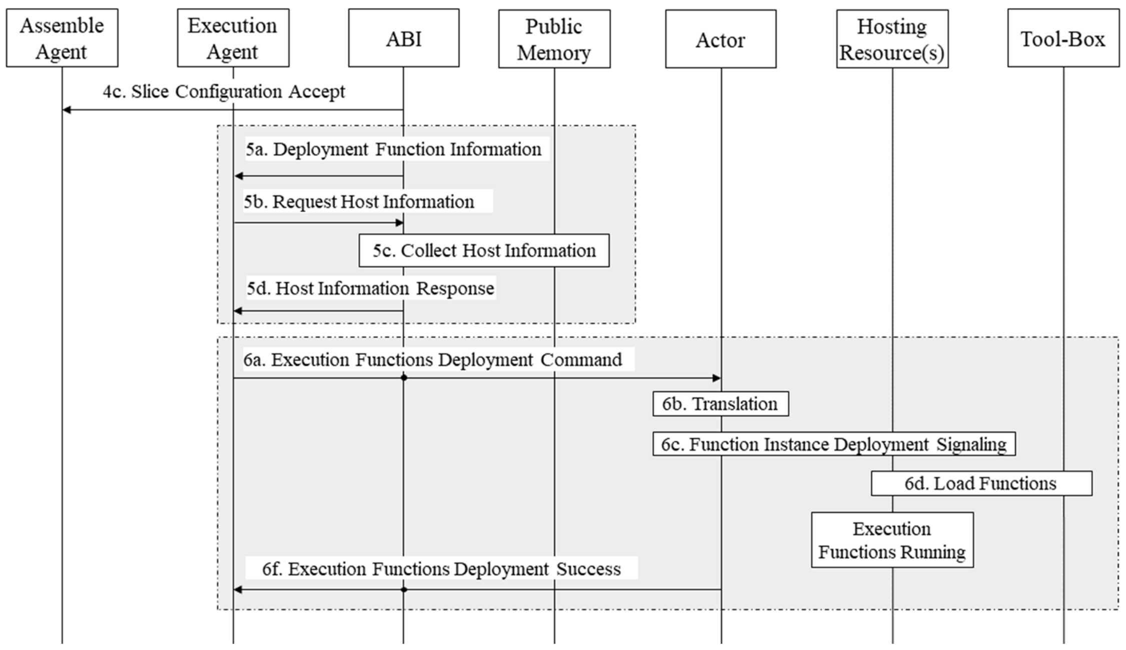
Assemble Agent
Execution Agent
ABI
Public Memory
Actor
Hosting Resource(s)
Tool-Box
4c. Slice Configuration Accept
5a. Deployment Function Information
5b. Request Host Information
5c. Collect Host Information
5d. Host Information Response
6a. Execution Functions Deployment Command
6b. Translation
6c. Function Instance Deployment Signaling
6d. Load Functions
Execution Functions Running
6f. Execution Functions Deployment Success

Figure 6.2-3: Deployment of the slice execution functions

The (high-level) procedure for the deployment of connection functions (following the deployment of execution functions on the hosting resources) is presented in Figure 6.2-4 below:

7) The ABI extracts the connection functions from the slice function list (determined by the Assemble Agent initially) to the Connection Agent. The connection agent decides which information is needed before the deployment of the functions, hence, requests the information about the RAN nodes involved from the Public Memory.

8) After the collection of UE and BS information from the Public Memory, the connection agent initiates the deployment of the connection functions through a command to the Actor. The Actor converts the message (e.g. from natural language) to RAN signalling towards BS and UEs.

NOTE 4: The functions to be deployed and run on the BS and UEs are assumed to be available in the RAN, therefore, the Toolbox is not involved here.

9) Once the connections functions are running, the ABI informs the Assemble and Planning agents about the successful deployment of the customized E2E slice.

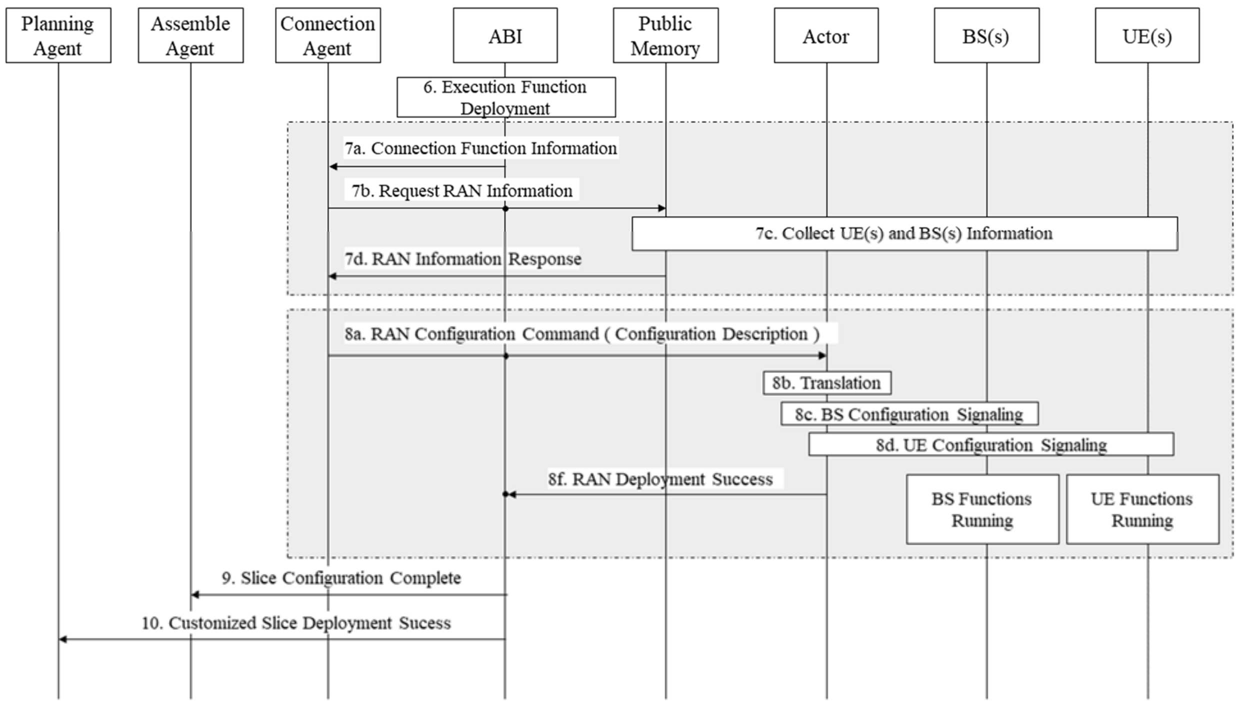
Planning Agent
Assemble Agent
Connection Agent
ABI
Public Memory
Actor
BS(s)
UE(s)
6. Execution Function Deployment
7a. Connection Function Information
7b. Request RAN Information
7c. Collect UE(s) and BS(s) Information
7d. RAN Information Response
8a. RAN Configuration Command (Configuration Description)
8b. Translation
8c. BS Configuration Signaling
8d. UE Configuration Signaling
8f. RAN Deployment Success
BS Functions Running
UE Functions Running
9. Slice Configuration Complete
10. Customized Slice Deployment Sucess

Figure 6.2-4: Procedure for the deployment of connection functions

The reconfiguration of the slice functions is presented in Figure 6.2-5:

1) After the slice deployment, the status is continuously monitored by data collection happening between the RAN, hosting resources, and the Public Memory.

2) If an unexpected status change is detected, the execution or connection agents (or both) are informed, respectively, depending on the actual event.

3) If the event requires the reconfiguration of the connection functions deployed (e.g. in a UE), the signalling is triggered via the Actor.

4) If the event relates to the execution functions (e.g. mobility management) deployed in hosting resources, then the execution function reconfigures these via the Actor.

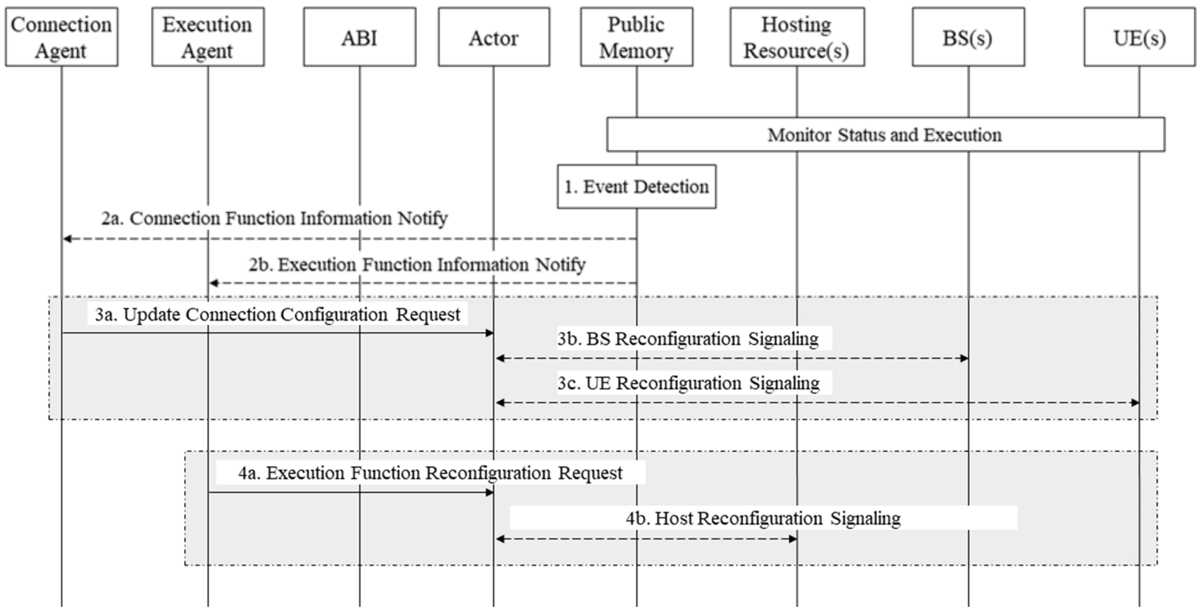
Connection Agent
Execution Agent
ABI
Actor
Public Memory
Hosting Resource(s)
BS(s)
UE(s)
Monitor Status and Execution
1. Event Detection
2a. Connection Function Information Notify
2b. Execution Function Information Notify
3a. Update Connection Configuration Request
3b. BS Reconfiguration Signaling
3c. UE Reconfiguration Signaling
4a. Execution Function Reconfiguration Request
4b. Host Reconfiguration Signaling

Figure 6.2-5: Slice function reconfiguration procedure

Lastly, the recycling of the E2E slice is shown in Figure 6.2-6:

1) The recycling procedure is initiated with the Actor receiving signalling messages that indicate request to free the network and computation resources.

2) Subsequently, the Connection Agent is involved to recycle the RAN connection functions involving e.g. UEs and BSs.

3) Next, the recycling procedure is continued with the goal to free the execution resources deployed on the hosting resources. This is performed through the interaction of Actor and the respective pool of computation resources.

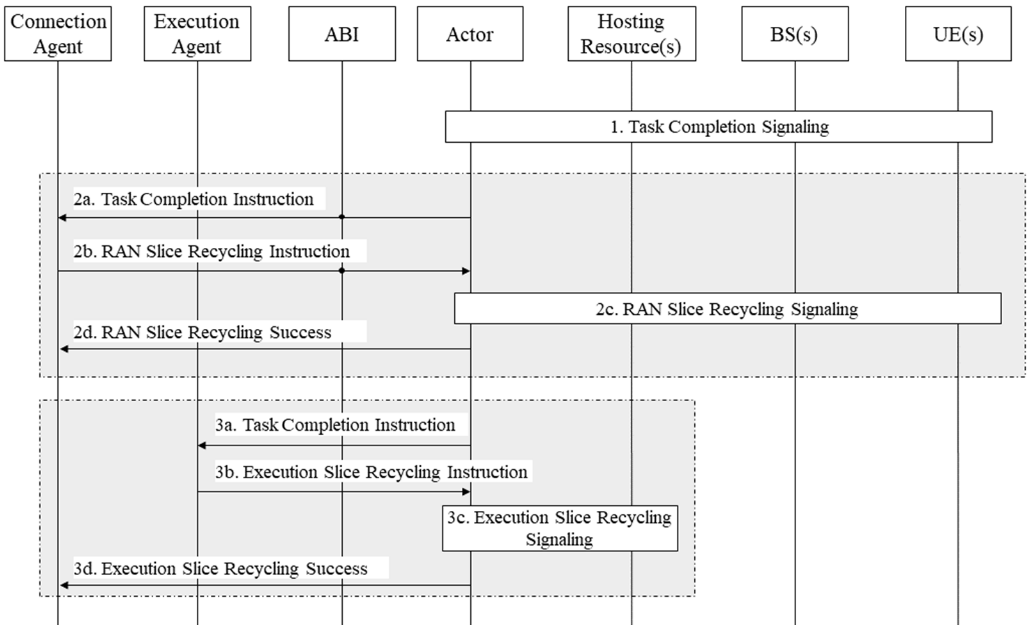
Connection Agent
Execution Agent
ABI
Actor
Hosting Resource(s)
BS(s)
UE(s)
1. Task Completion Signaling
2a. Task Completion Instruction
2b. RAN Slice Recycling Instruction
2c. RAN Slice Recycling Signaling
2d. RAN Slice Recycling Success
3a. Task Completion Instruction
3b. Execution Slice Recycling Instruction
3c. Execution Slice Recycling Signaling
3d. Execution Slice Recycling Success

Figure 6.2-6

7 Conclusion and Recommendations

7.1 Conclusions

The next-generation network slicing requires a flexible assembly of a set of capabilities and multi-dimensional resources and autonomous adaptation to the changing network environment and customer demands. To support advanced network slicing, a new AI Agent based Core network (AI-Core) is proposed. It can create network slices based on the customer's high-level requirement, autonomously and intelligently control network slices to adapt to the dynamic wireless network environment, and learn from the network feedback to optimize itself and the slices it creates over time. Increased network autonomy is expected to be achieved through the help of AI-Core.

7.2 Recommendations

The following items are for further study:

1) Examine how best to implement a RAG system (as specified in clauses 4.2.1, 4.3.1 and others). The main issues are: (1) should it be defined internal to AgentGPT or not, and (2) does NetGPT have its own dedicated RAG system.

2) How does the Planning Agent perform its task defined in clause 4.3.2? Specifically, can it reuse technologies from [i.5], [i.29], and [i.36] ?

3) Examine Memory and its relationship to different memory functional blocks as defined in [i.29].

4) Examine if and how a foundation model should be created and trained that is further used to produce a set of fine-tuned models for domain-specific tasks. A promising approach would be to create a slicing-specific Foundation Model from the original Foundation Model for classic use cases such as eMBB and uRLLC, while other use cases (e.g. Smart City and eHealth) could be created directly from the original Foundation Model.

5) Examine if an AI Agent acting as an orchestrator is needed to improve performance.

6) Examine how best to support imperative, declarative, and intent-based interactions.

7) Examine if there should be a feedback loop from the Sandbox tests to a collective memory.

8) Review communication interfaces and recommend a set for the ABI.

9) Examine and document a recommended approach for implementing NetGPT. In particular, choose from the two approaches in clause 4.4 (training from scratch vs. finetuning), as well as other approaches not mentioned in the present document if applicable.

10) Examine which mechanisms can be used to augment the capabilities of NetGPT and each of the AgentGPT entities. It is recommended to refer to [i.5] and its references for this task.

11) Examine if it is desirable for the AI-Core to interact with an ENI System. The simplest interaction is to through APIs using the API Broker of the ENI System. However, other interactions are possible.

NOTE: This is probably best done as a separate document.

History

Document history		
V4.1.1	February 2025	Publication