

Architecture for Self-Evolution of 6G Core Network

Based on Intelligent Decision Making

(Point-by-Point Response)

Lu Lu, Chao Liu, Chunhong Zhang, Zheng Hu, Shangjing Lin, Zihao Liu, Meng Zhang, Xinshu Liu, Jinhao Chen

Response to Reviewer 3 Comments

We sincerely appreciate the valuable suggestions you provided. Based on your suggestions, we have made the necessary modifications to the manuscript. To ensure clarity, we have highlighted the changes using a gentle yellow color. Additionally, in the subsequent sections, we have provided point-by-point responses to address each of your comments.

Point 1: Authors are suggested to reduce the introduction section currently it is very long. The introduction section should be concise and to the point. It comprised of summarization of recent studies, your contribution, the benefits of your research, and your motivation also.

Response 1: Thank you for your valuable suggestion. Following your advice, we have carefully streamlined the content, resulting in a more concise and focused introduction. The revised version now effectively presents the key aspects while maintaining clarity and coherence.

Point 2: The paper organization is missing at the end of the introduction section. For example; in a section we reviewed the state of the art etc.

Response 2: Following your suggestion, we have now added paper organization at the end of the Introduction. This new part provides a comprehensive overview of the paper's structure, outlining the key points covered in each section. The revised version is as follows:

Paper Organization: The rest of this paper is organized as follows. In Section 2, we introduce the evolution of core network architecture and discuss relevant works on Network Intelligence. In Section 3, we present the Problem Formulation for Network Self-Evolution. In Section 4, we propose the Self-Evolving Pipeline and a deployment architecture for intelligent agents in the 6G core network. In Section 5, we present the details of our algorithm. Section 6 is dedicated to the validation of our algorithm, where we conduct experiments to assess its performance. Finally, Section 7 is used to conclude our research.

Point 3: Figure 6 explains the architecture of your proposed study. It is requested to explain in detail.

Response 3: To clarify the meaning of Figure 6, in addition to the overall two-layer architecture of the self-evolution pipeline we have already discussed in the last version, we further add detailed descriptions on 3 points as follows:

1. we added a use case for Knowledge Graph assisted decision making in the CCN, constructing a knowledge graph with the entities and the relationships containing knowledge of 3GPP protocols, Cloud Evolution Agent and statistics of the CCN to reduce the decision space of the CCN evolution.

2. We add the details of the Edge scenario data by incorporating additional information, such as the signaling throughput, the distribution of various signaling types, and quality of service (QoS) of UEs.

3. We add details on CCN evolution. The significant changes in the architecture of numerous ECNs will result in changes to the environment of the CCN, such as signaling throughput of ECN and signaling distribution of ECN. Changes in environment will result in alterations to the composition of microservices, thereby evolve the CCN.

The revised Section 4.2:

At the top-layer of the architecture, the central CCN deploys AI models as its cloud evolving agent. This cloud agent is responsible for gathering information about itself and all ECNs to learn the evolving policy for the CCN. It creates training data and updates its evolving policy to maximize rewards, ensuring that the CCN evolves in line with its agent's decisions. To optimize resource sharing and reuse the microservices among all ECNs, it is feasible to deploy only a single global gene library Glib in the CCN. Glib includes microservices derived from network decomposition as well as manually added microservices. The CCN's agent is responsible for creating, maintaining, and updating the Glib to ensure that all the ECNs can efficiently access and utilize the shared microservices. By doing so, the self-evolving architecture can achieve greater scalability and consistence while minimizing resource consumption. To enhance the intelligence of the cloud agent and ensure automatic maintenance of Glib, a knowledge graph is often deployed in the CCN, which incorporates domain expert prior knowledge and evolving preferences of 6G service operators. This allows the cloud agent to learn from the experts' insights and make informed decisions about how to evolve the network. By incorporating this knowledge into its evolving policy, the cloud agent can improve its performance and help the CCN evolve more efficiently. **For instance, we can construct a knowledge graph with the entities and the relationships containing knowledge of 3GPP protocols, Cloud Evolution Agent and statistics of the CCN to reduce the decision space of the CCN evolution.**

At the bottom-layer of the architecture, there are multiple ECN designed for various application scenarios. Each ECN is equipped with its own evolving agent, which receives edge data **such as the signaling throughput, the distribution of various signaling types, and quality of service (QoS) of UEs.** and is responsible for making decisions about how to evolve its host ECN. Specifically, the edge agent learns how to use the genes from the global gene library Glib, residing on the CCN, to compose desirable network function NFs and then, the edge core network structure. As the application scenario of the ECN changes, as shown in Figure 6, where the applications expand from single autonomous vehicles to a mixture of services including autonomous vehicles, video surveillance, and Internet of Things (IoT), the agent decides to reshape the structure of the

ECN from two NFs into three NFs, while also altering the internal microservice compositions of each NF. In the long-term, the evolving policies of different agents will be highly distinct due to their ECNs serving different application scenarios. As a result, the architecture of ECNs will also vary accordingly. The significant changes in the architecture of numerous ECNs will result in changes to the environment of the CCN, such as signaling throughput of ECN and signaling distribution of ECN. Changes in environment will result in alterations to the composition of microservices, thereby evolve the CCN.

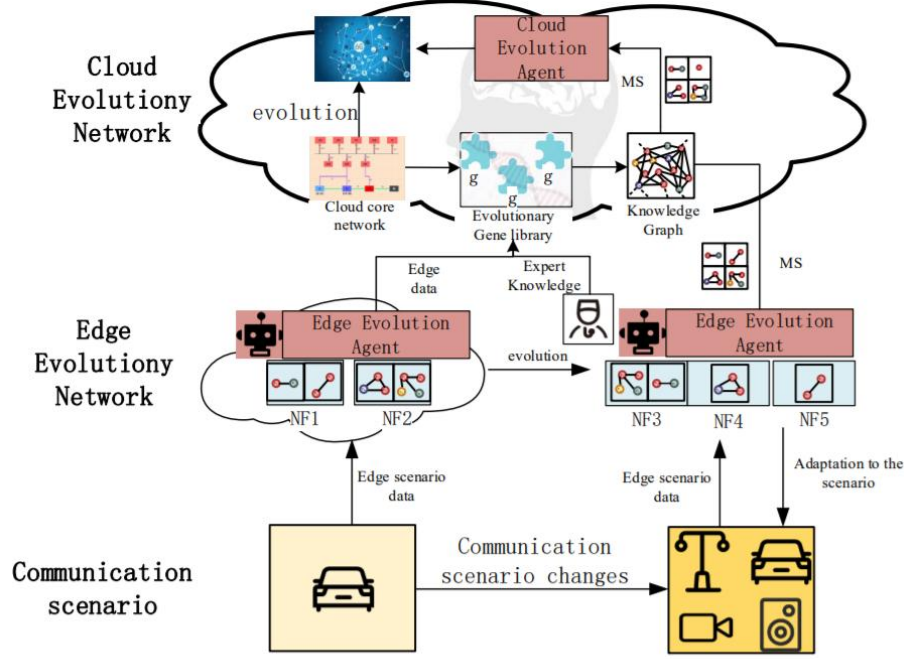


Figure 6. The hierarchical architecture of self-evolution pipeline for 6G Core Network.

Point 4: The figure size in the paper are not equal. It should be equal in the entire manuscript.

Response 4: Thank you for the valuable feedback regarding the image sizes. We have now adjusted all images, including Figure 2, to have uniform sizes, ensuring a more consistent and visually appealing layout throughout the manuscript.

Point 5: Algorithm 1 explanation is required. Especially the parameters that are used in this figure.

Response 5: Sorry for not providing a detailed description of the algorithm 1's parameters, which has caused trouble to you. We have added a detailed part (the last paragraph of Section 5.3) before the Pseudocode of Algorithm 1, hoping it can help you understand the concept of this algorithm.

The last paragraph of Section 5.3:

The pseudocode of the Self-Evolving algorithm is presented as bellow. The algorithm takes the initial strategy π_E , network elements NF_{t0} , Gene pool $GLib_{t0}$ and the initial state of the environment E_{t0} as inputs, and then outputs an update strategy $\pi_E = \pi_{E0}, \pi_{E1}, \pi_{E2}$ after

completing three stages of training. In the first stage $\varphi_t = 0$, the self-evolving agent samples action A_t^G to split microservices and update the gene pool $GLib_t$ through policy π_{E0} . In the second stage $\varphi_t = 1$, the agent samples action A_t^C to reorganize microservices to form NF_t and topologies through policy π_{E1} . In the third stage $\varphi_t = 2$, the agent samples action A_t^E to generate dynamic service chains and interact with the environment E_t to return reward function r_t through policy π_{E2} . All parameters are stored in memory pool M , and we use Proximal Policy Optimization(PPO) to update the strategy.

including CPU occupancy, time latency and specific rewards for different tasks.

The pseudocode of the Self-Evolving algorithm is presented as bellow. The algorithm takes the initial strategy π_E , network elements NF_{t0} , Gene pool $GLib_{t0}$ and the initial state

Algorithm 1: Self-Evolving algorithm

of the environment E_{t0} as inputs, and then outputs an update strategy $\pi_E = \pi_{E0}, \pi_{E1}, \pi_{E2}$ after completing three stages of training. In the first stage $\varphi_t = 0$, the self-evolving agent samples action A_t^G to split microservices and update the gene pool $GLib_t$ through policy π_{E0} . In the second stage $\varphi_t = 1$, the agent samples action A_t^C to reorganize microservices to form NF_t and topologies through policy π_{E1} . In the third stage $\varphi_t = 2$, the agent samples action A_t^E to generate dynamic service chains and interact with the environment E_t to return reward function r_t through policy π_{E2} . All parameters are stored in memory pool M , and we use Proximal Policy Optimization to update the strategy.

Point 6: Did you compare your results with the state of the art? If yes, please mention it in figures

Response 6: We appreciate the inquiry. In this study, we did not compare our algorithm with SOTA directly. The reason is as follow. According to our knowledge, the existing research works [1,2] on network architecture evolution mainly focus on pre-configured approaches. It is totally different in nature from our proposed method, which allows the core network to evolve itself dynamically by decomposing and recomposing the network functions. Thus, there is no comparable SOTA work to our algorithm. Instead, to demonstrate the efficiency of our architecture, the experimental comparison is made the performance between the pre-evolution network and the post-evolution network in terms of latency and resource consumption. The performance improvement of the post-evolution network clearly indicates that the self-evolution strategy can enhance the adaptability of the 6G core network to the changing environment.

Reference:

- [1] Trakadas, Panagiotis, et al. "A cost-efficient 5G non-public network architectural approach: Key concepts and enablers, building blocks and potential use cases." *Sensors* 21.16 (2021): 5578.
- [2] Rostami, Ahmad. "Private 5G networks for vertical industries: Deployment and operation models." 2019 IEEE 2nd 5G World Forum (5GWF). IEEE, 2019.

Point 7: The detail of simulation environment should be mention in the results section

Response 7: We add details about the hardware and the Gym[59] environment used in our simulation environment. First, we used an NVIDIA TITAN 1080 graphics card and two Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors (8 cores each) for our simulation environment. Second, In the simulation, we utilized OpenAI Gym, an open-source Python library for reinforcement learning algorithms, and developed our self-evolving environment. It consists of user equipment (UE), microservices, and network functions (NFs). The UE serves as a signaling transceiver. Our microservices and NFs were inspired by free5gc's[54] architecture but without intermediate processing steps, focusing solely on calculating performance metrics during user interactions. For instance, when a user connects to the Access and Mobility Management Function (AMF), we skip detailed signaling procedures and directly calculate processing delay and CPU usage.

Reference:

[54] Free5GC. Available online: <https://www.free5gc.org/> (accessed on 18 May 2023).

[59] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

The added Section 6.2:

~~We conducted self-evolution experiments using a simulation environment called Gym[56], which is an open-source platform providing a set of tools and algorithms for creating, evaluating, and optimizing reinforcement learning (RL) environments.~~

We used an NVIDIA TITAN 1080 graphics card and two Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors (8 cores each) for our experiments. We conducted self-evolution experiments in the Gym[59] simulation environment, which is an open-source platform providing a set of tools and algorithms for creating, evaluating, and optimizing reinforcement learning (RL) environments. To overcome Gym's lack of a dedicated self-evolution environment setting, we developed a custom simulation environment with user equipment(UE), microservices and network functions(NF) for self-evolution experiments. The UE serves as a mere signaling transceiver within the system. In addition, the design of our microservices and NFs draws inspiration from free5gc's architecture. However, in contrast to free5gc, we have omitted the intermediate processing steps in our microservices and network functions. Instead, we focus solely on calculating performance during user interactions.

Point 8: The future work should be mention in the conclusion section

Response 8: We add the future work in the conclusion section.

The added Section 6.3: For future work, we consider three points. First, we have discussed the evolving approaches for the Function F and Topology T of a SCN = {F, P, T}, however, we have not

explicitly addressed the protocol P. When the network functions NFs are reconstructed, the corresponding core network protocols, **which can be regarded as a conversation between NFs**, would also be automatically regenerated. The AIGC (AI Generated Content) provides potential mechanisms for this aspect. ~~Second, the physical placement of the evolved NF instances would also be considered as an individual stage of the pipeline to optimize the core network.~~ **Second, considering the physical or virtual environment of the edge core network, the physical deployment of evolved network function instances should also be studied.** Finally, the multi-agent evolution algorithm for the joint-optimization for the Cloud Core Network and Edge Core Network would be studied.