GNN-Based Multi-Agent RL for Dynamic AP-CPU Assignment in Cell-Free Massive MIMO

Mahnoor Ajmal, Joohwan Park, Dongkyun Kim
School of Computer Science and Engineering, Kyungpook National University, Daegu, Republic of Korea
Emails: {mahnoor.ajmal, drosis, dongkyun}@knu.ac.kr

***Abstract*—The deployment of cell-free massive multiple-input multiple-output (CF-mMIMO) systems at scale requires distributed processing across multiple central processing units (CPUs). The assignment of access points (APs) to CPUs directly impacts computational load distribution and network performance. Existing optimization methods rely on global channel knowledge and iterative solvers, limiting real-time applicability. This paper presents a graph neural network (GNN) framework with centralized training and decentralized execution (CTDE) for AP-CPU assignment. The network topology is represented as a graph where APs form nodes and edges capture interference relationships. Graph attention layers generate embeddings that enable each AP to select its CPU assignment through a sequential coordination mechanism. Simulation results for a network with 50 APs and 20 users demonstrate 78% reduction in CPU load imbalance compared to distance-based assignment, while maintaining equivalent throughput. The learned policy achieves the highest minimum user rate of 10.82 bps/Hz among all methods. Additional scalability experiments across 30 to 70 APs confirm consistent improvements of 73--82%, validating the effectiveness of learning-based approaches for backhaul resource management.**

***Index Terms*—Cell-free massive MIMO, GNN, multi-agent reinforcement learning, AP-CPU assignment, load balancing**

I. Introduction

Cell-free massive multiple-input multiple-output (CF-mMIMO) is widely recognized as a pivotal enabler for upcoming sixth-generation (6G) communication systems [1]. By discarding traditional cellular boundaries, this architecture leverages a dense array of geographically distributed access points (APs) to provide joint service to all users in the network. These APs maintain wireless fronthaul connections with users while exchanging data with a central processing unit (CPU) through wired or wireless backhaul infrastructure. Such a distributed topology effectively mitigates the cell-edge phenomenon, ensuring consistent spectral efficiency throughout the coverage zone [2].

The original CF-mMIMO framework assumed that all APs are connected to a single CPU, which performs centralized signal processing for the entire network. While theoretically appealing, this design suffers from poor scalability. As the number of users and APs grows, the computational burden at the CPU increases substantially, and the backhaul capacity becomes a bottleneck. To mitigate this, user-centric approaches were introduced where each user is served by only a subset of nearby APs rather than the full network [3]. Although user-centric clustering reduces fronthaul overhead, the single-CPU architecture remains a fundamental limitation for large-scale deployments.

To enable truly scalable CF-mMIMO, recent works have considered architectures with multiple CPUs distributed across the network [4], [5]. In such systems, each AP must be assigned to exactly one CPU for signal processing. This AP-to-CPU assignment directly affects both the computational load distribution among CPUs and the achievable network throughput. An unbalanced assignment where some CPUs handle many more APs than others leads to processing delays at overloaded CPUs and underutilization of others. Finding the optimal assignment that balances CPU loads while maximizing throughput is therefore critical for practical multi-CPU CF-mMIMO systems.

Existing approaches to the AP-CPU assignment problem can be broadly categorized into optimization-based methods and learning-based methods. In context of optimization-based approaches, authors in [5] address the joint optimization of fronthaul loading and resource allocation by modeling it as a mixed-integer linear program (MILP). They proposed a successive convex approximation (SCA) algorithm that demonstrated improved network performance through explicit traffic management. Dynamic cooperative clustering (DCC) frameworks have also been developed using combinatorial algorithms such as the Kuhn-Munkres method to match APs with CPUs based on traffic statistics and backhaul capacity [2], [4]. While these optimization-based methods provide strong performance guarantees, they rely on iterative solvers with high computational complexity. The requirement for global channel state information (CSI) and the slow convergence of iterative algorithms make these approaches impractical for real-time adaptation in time-varying wireless environments.

To address the computational limitations of optimization methods, deep reinforcement learning (DRL) has been applied to CF-mMIMO resource allocation. Authors in [6] proposed a distributed DRL framework for AP clustering, where each user is associated with a virtual CPU that dynamically selects serving APs. This approach reduces signaling overhead compared to centralized methods. However, the virtual CPU concept focuses on logical user-centric clusters and does not explicitly model the physical constraints of multi-CPU hardware. The competition among APs for limited CPU processing capacity and backhaul bandwidth is not captured in this formulation.

Recently, Graph Neural Networks (GNNs) have gained prominence as a powerful solution for managing wireless

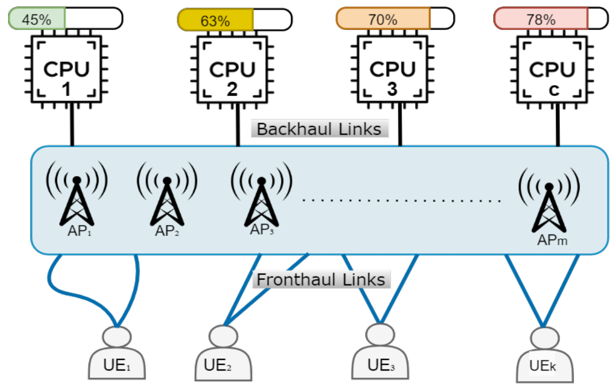


Fig. 1. Cell-free massive MIMO with multiple CPUs

resources, largely because they excel at capturing the inherent topological structure of networks [7], [8]. By modeling APs and their spatial relationships as a graph, GNNs can capture interference patterns and proximity information that feed-forward networks cannot. Despite the success of GNNs in power control and beamforming problems, their application to AP-CPU assignment in multi-CPU CF-mMIMO remains unexplored.

A. *Contributions*

This paper proposes a graph neural network framework with centralized training and decentralized execution (GNN-CTDE) for dynamic AP-to-CPU assignment in multi-CPU CF-mMIMO systems. The proposed approach addresses the limitations of both optimization-based and existing learning-based methods by combining the representational power of GNNs with the scalability of multi-agent reinforcement learning. The main contributions are summarized as follows:

- We formulate the AP-CPU assignment as a cooperative multi-agent problem where each AP acts as an independent agent that selects its CPU assignment. A graph attention network (GAT) encoder captures the spatial topology and interference relationships among APs, generating node embeddings that inform each agent's decision.
- We introduce a sequential action selection mechanism that enables implicit coordination among decentralized agents. Each agent observes the accumulated CPU load from prior assignments when making its decision, allowing agents to naturally avoid overloaded CPUs without explicit inter-agent communication during execution.
- We design a reward structure that combines global network objectives with local credit assignment. The global component captures sum-rate throughput, user fairness, and CPU load balance, while the local component provides each agent with feedback on its individual contribution to load balancing.
- Numerical results show that the proposed GNN-CTDE achieves 78% reduction in CPU load balancing compared to distance-based assignment while maintaining equivalent throughput.

II. System Model and Problem Formulation

A. *Network Architecture*

Consider a cell-free massive MIMO network deployed over a square area of $S \times S$ meters, as shown in Fig. 1. The network comprises M APs, K single-antenna users, and C central processing units (CPUs). Each AP is equipped with L antennas.

The network has two distinct link types. On the fronthaul, APs communicate with users through wireless channels. On the backhaul, each AP connects to exactly one CPU through capacity-constrained wireless links. The CPUs handle baseband processing tasks including channel estimation, precoding computation, and signal detection. This work focuses on the backhaul segment, specifically the assignment of APs to CPUs.

Let $\mathcal{M} = \{1, \ldots, M\}$, $\mathcal{K} = \{1, \ldots, K\}$, and $\mathcal{C} = \{1, \ldots, C\}$ denote the sets of APs, Users, and CPUs. The AP-to-CPU assignment is defined by binary variables

$$a_{m,c} = \begin{cases} 1, & \text{if AP } m \text{ connects to CPU } c \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

with the constraint that each AP connects to exactly one CPU:

$$\sum_{c=1}^{C} a_{m,c} = 1, \quad \forall m \in \mathcal{M}. \quad (2)$$

B. *Backhaul Model and CPU Load*

Each CPU processes the signals from all APs assigned to it. The computational load at CPU c is proportional to the number of connected APs:

$$L_c = \sum_{m=1}^{M} a_{m,c}. \quad (3)$$

The processing latency at a CPU increases with its load. When loads are unbalanced, overloaded CPUs become bottlenecks that degrade network performance, while underloaded CPUs waste resources.

We impose a capacity constraint $L_c \leq L_{\max}$ to prevent any CPU from exceeding its processing capability. The load imbalance across the network is quantified by the standard deviation:

$$\sigma_L = \sqrt{\frac{1}{C} \sum_{c=1}^{C} (L_c - \bar{L})^2}, \quad (4)$$

where $\bar{L} = M/C$ is the ideal balanced load. A well-designed assignment achieves $\sigma_L \approx 0$.

The backhaul also affects coordination among CPUs. Following user-centric clustering [9], each UE k is served by a subset $\mathcal{M}_k$ of D nearby APs. When these serving APs connect to different CPUs, inter-CPU coordination is required on the backhaul to exchange precoding information. Let

$$N_k^{\text{CPU}} = |\{\phi(m) : m \in \mathcal{M}_k\}| \quad (5)$$

denote the number of distinct CPUs serving UE k. Higher values of N_k^{CPU} increase backhaul signaling overhead.

C. Channel Model and Spectral Efficiency

The channel between AP m and UE k is modeled as $\mathbf{h}_{mk} = \sqrt{\beta_{mk}}\tilde{\mathbf{h}}_{mk}$, where β_{mk} captures large-scale fading and $\tilde{\mathbf{h}}_{mk} \sim \mathcal{CN}(\mathbf{0}, \mathbf{I}_L)$ represents small-scale fading. Using maximum ratio transmission and standard capacity bounds [9], the spectral efficiency of UE k is

$$R_k = \log_2(1 + \text{SINR}_k). \tag{6}$$

The sum-rate is $R_{\text{sum}} = \sum_{k=1}^{K} R_k$ and the minimum rate is $R_{\min} = \min_k R_k$.

D. Problem Formulation

The AP-CPU assignment problem seeks a mapping that maximizes throughput while balancing CPU loads:

$$\max_{\{a_{m,c}\}} \quad \alpha_1 R_{\text{sum}} + \alpha_2 R_{\min} - \alpha_3 \sigma_L \tag{7a}$$

$$\text{s.t.} \quad \sum_{c=1}^{C} a_{m,c} = 1, \quad \forall m \tag{7b}$$

$$L_c \leq L_{\max}, \quad \forall c \tag{7c}$$

$$a_{m,c} \in \{0, 1\}, \quad \forall m, c \tag{7d}$$

where $\alpha_1, \alpha_2, \alpha_3$ weight the objectives. This mixed-integer problem has C^M possible assignments, making exhaustive search infeasible. We propose a learning-based approach in the next section.

III. Proposed GNN-CTDE Framework

This section presents the graph neural network framework with centralized training and decentralized execution (GNN-CTDE) for solving the AP-CPU assignment problem. Fig. 2. illustrates the overall architecture.

A. Centralized Training, Decentralized Execution

The CTDE paradigm separates the training and deployment phases. During training, a central controller has access to global network information and coordinates the learning of all agents. This central controller can reside on any server with sufficient computational resources and is used only offline. During deployment, each AP acts independently using only local observations, requiring no real-time communication with a central entity.

This separation addresses a key challenge in multi-CPU systems. While training benefits from global coordination to learn effective policies, practical deployment requires distributed decision-making since real-time centralized control would introduce unacceptable latency. In our framework, the training server collects experiences from the network, updates all agent policies simultaneously, and distributes the learned parameters to individual APs. Once deployed, each AP makes CPU selection decisions locally without consulting the training server or other APs.

B. Graph Representation

We model the cell-free network as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where nodes $\mathcal{V}$ represent APs and edges $\mathcal{E}$ capture interference relationships. An edge connects APs m and m' if they serve at least one common user, indicating potential interference coupling:

$$(m, m') \in \mathcal{E} \iff \mathcal{K}_m \cap \mathcal{K}_{m'} \neq \emptyset. \tag{8}$$

Each node m has a feature vector $\mathbf{x}_m \in \mathbb{R}^F$ containing:

- Channel statistics: Average and minimum SINR to served users
- Load information: Current assignment and CPU distances
- Position: Normalized AP coordinates

This graph structure allows the GNN to learn spatial patterns and interference relationships that influence assignment quality.

C. Graph Attention Encoder

A graph attention network (GAT) processes $\mathcal{G}$ to produce node embeddings. For each AP m, the encoder aggregates information from neighboring APs using attention weights:

$$\mathbf{z}_m = \sigma\left(\sum_{m' \in \mathcal{N}(m)} \alpha_{mm'} \mathbf{W}\mathbf{x}_{m'}\right), \tag{9}$$

where $\mathcal{N}(m)$ is the neighborhood of AP m, $\mathbf{W}$ is a learnable weight matrix, and $\alpha_{mm'}$ are attention coefficients computed as

$$\alpha_{mm'} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{x}_m \| \mathbf{W}\mathbf{x}_{m'}]))}{\sum_{j \in \mathcal{N}(m)} \exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{x}_m \| \mathbf{W}\mathbf{x}_j]))}. \tag{10}$$

The encoder uses two GAT layers with multi-head attention, producing embeddings $\mathbf{z}_m \in \mathbb{R}^{32}$ that capture both local features and network-wide context.

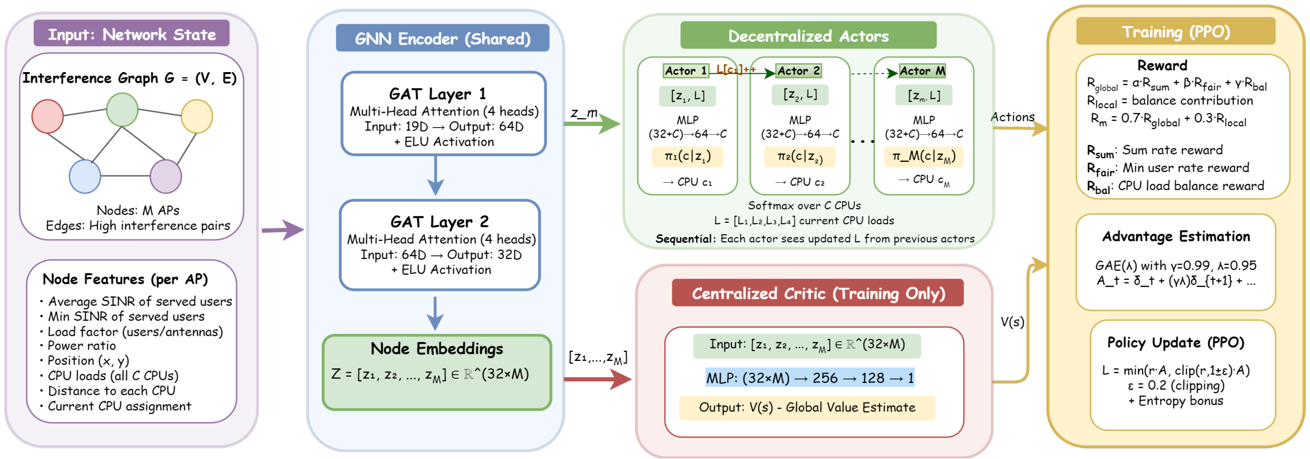
D. Sequential Action Selection

A naive approach would have all APs select CPUs simultaneously. However, this leads to poor coordination since no agent knows what others will choose. We introduce sequential selection where agents act one after another, each observing the accumulated load from previous decisions.

At each decision round, agents are processed in sequence. Agent m receives its embedding $\mathbf{z}_m$ and the current load vector $\mathbf{L} = [L_1, \ldots, L_C]$ reflecting assignments made by earlier agents. The policy network outputs a distribution over CPUs:

$$\pi_m(c|\mathbf{z}_m, \mathbf{L}) = \text{softmax}(f_\theta([\mathbf{z}_m; \mathbf{L}/L_{\max}])), \tag{11}$$

where f_θ is a two-layer neural network. By conditioning on $\mathbf{L}$, later agents can avoid CPUs that earlier agents have filled, achieving implicit coordination without explicit communication.



Input: Network State
Interference Graph G = (V, E)
Nodes: M APs
Edges: High interference pairs
Node Features (per AP)
• Average SINR of served users
• Min SINR of served users
• Load factor (users/antennas)
• Power ratio
• Position (x, y)
• CPU loads (all C CPUs)
• Distance to each CPU
• Current CPU assignment
GNN Encoder (Shared)
GAT Layer 1
Multi-Head Attention (4 heads)
Input: 19D → Output: 64D
+ ELU Activation
GAT Layer 2
Multi-Head Attention (4 heads)
Input: 64D → Output: 32D
+ ELU Activation
Node Embeddings
Z = [z₁, z₂, ..., z_M] ∈ ℝ^(32×M)
z_m
[z₁,...,z_M]
Decentralized Actors
Actor 1
L[c₁]++
Actor 2
Actor M
[z₁, L]
[z₂, L]
[z_m, L]
MLP
(32+C)→64→C
π₁(c|z₁)
π₂(c|z₂)
π_M(c|z_M)
→ CPU c₁
→ CPU c₂
→ CPU c_M
Softmax over C CPUs
L = [L₁,L₂,L₃,L₄] current CPU loads
Sequential: Each actor sees updated L from previous actors
Actions
Centralized Critic (Training Only)
Input: [z₁, z₂, ..., z_M] ∈ ℝ^(32×M)
MLP: (32×M) → 256 → 128 → 1
Output: V(s) - Global Value Estimate
V(s)
Training (PPO)
Reward
R_global = α·R_sum + β·R_fair + γ·R_bal
R_local = balance contribution
R_m = 0.7·R_global + 0.3·R_local
R_sum: Sum rate reward
R_fair: Min user rate reward
R_bal: CPU load balance reward
Advantage Estimation
GAE(λ) with γ=0.99, λ=0.95
A_t = δ_t + (γλ)δ_{t+1} + ...
Policy Update (PPO)
L = min(r·A, clip(r,1±ε)·A)
ε = 0.2 (clipping)
+ Entropy bonus

Fig. 2. GNN-CTDE architecture. The GAT encoder processes the network graph to produce AP embeddings. Each AP agent selects a CPU based on its embedding and observed loads.

E. Reward Design

The reward combines global objectives with local credit assignment. The global component is

$$r_{\text{global}} = w_1 \tilde{R}_{\text{sum}} + w_2 \tilde{R}_{\text{min}} + w_3 r_{\text{balance}} + w_4 r_{\text{coord}}, \tag{12}$$

where $\tilde{R}_{\text{sum}}$ and $\tilde{R}_{\text{min}}$ are normalized throughput terms, r_{balance} penalizes load imbalance, and r_{coord} encourages multi-CPU coordination per user.

To address credit assignment in multi-agent learning, each agent also receives a local reward based on its individual contribution:

$$r_m = 0.7 \cdot r_{\text{global}} + 0.3 \cdot r_m^{\text{local}}, \tag{13}$$

where r_m^{local} rewards selecting underloaded CPUs and penalizes selecting overloaded ones.

F. Training with PPO

We train the framework using Proximal Policy Optimization (PPO) [10]. A centralized critic estimates the value function from the joint embedding of all APs:

$$V(\mathbf{s}) = g_\phi([\mathbf{z}_1; \mathbf{z}_2; \dots; \mathbf{z}_M]), \tag{14}$$

where g_ϕ is a neural network. The critic is used only during training to compute advantages for policy updates. During deployment, only the distributed actor policies are needed.

The complete GNN-CTDE procedure is summarized in Algorithm 1.

During training (lines 3-19), the framework collects experiences through sequential action selection where each agent observes accumulated loads before deciding. The critic provides value estimates for advantage computation while actors update via clipped surrogate objective. At deployment (lines 25-30), only forward passes through the trained networks are required, enabling real-time execution without iterative optimization.

Algorithm 1 GNN-CTDE for AP-CPU Assignment

```
Require: APs $M$, Users $K$, CPUs $C$, Episodes $E$, Steps $T$
Ensure: Trained policy networks $\{\pi_m\}_{m=1}^M$
1:  Initialize: GAT encoder $\mathcal{G}_\theta$, actor networks $\{\pi_m\}$, critic $V_\phi$
2:  // Centralized Training Phase
3:  for episode $= 1$ to $E$ do
4:      Reset environment and trajectory buffer $\mathcal{B}$
5:      for $t = 1$ to $T$ do
6:          $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_M] \leftarrow \mathcal{G}_\theta(\mathcal{G})$      ▷ GNN encoding
7:          $\mathbf{L} \leftarrow [0, \dots, 0]^C$      ▷ Initialize CPU loads
8:          for $m = 1$ to $M$ do      ▷ Sequential selection
9:              Sample $c_m \sim \pi_m(\cdot|\mathbf{z}_m, \mathbf{L})$
10:             $L_{c_m} \leftarrow L_{c_m} + 1$
11:             Store $(\mathbf{z}_m, \mathbf{L}, c_m, \log \pi_m(c_m))$ in $\mathcal{B}$
12:         end for
13:         Apply assignment $\mathbf{c} = [c_1, \dots, c_M]$
14:         Observe $R_{\text{sum}}$, $R_{\text{min}}$, $\sigma_L$ from environment
15:         Compute $r_{\text{global}}$ using (12)
16:         for $m = 1$ to $M$ do
17:             $r_m \leftarrow 0.7 \cdot r_{\text{global}} + 0.3 \cdot r_m^{\text{local}}$
18:         end for
19:     end for
20:     Compute advantages $\hat{A}_m^t$ via GAE($\gamma$, $\lambda$)
21:     Update $V_\phi$ by minimizing $(V_\phi(\mathbf{s}) - R^{\text{target}})^2$
22:     Update $\pi_m$ via PPO: $\max \min(\rho\hat{A}, \text{clip}(\rho, 1 \pm \epsilon)\hat{A})$
23: end for
24: // Decentralized Execution Phase
25: $\mathbf{Z} \leftarrow \mathcal{G}_\theta(\mathcal{G})$, $\mathbf{L} \leftarrow \mathbf{0}$
26: for $m = 1$ to $M$ do
27:     $c_m \leftarrow \arg\max_c \pi_m(c|\mathbf{z}_m, \mathbf{L})$
28:     $L_{c_m} \leftarrow L_{c_m} + 1$
29: end for
30: return $\{c_1, c_2, \dots, c_M\}$
```

TABLE I
SIMULATION PARAMETERS

Parameter	Value
Number of APs (M)	50
Number of users (K)	20
Number of CPUs (C)	4
Antennas per AP (L)	4
Serving APs per user (D)	4
Coverage area	500×500 m^2
Carrier frequency	2 GHz
Bandwidth	20 MHz
AP transmit power	200 mW
Noise power	-94 dBm
Shadow fading std.	8 dB
Training episodes	500
Learning rate	5×10^{-5}
Discount factor (γ)	0.99
PPO clip (ϵ)	0.2
GNN embedding dimension	32

IV. NUMERICAL RESULTS

This section evaluates the proposed GNN-CTDE framework through simulations. We first describe the simulation setup and baseline methods, then present the main performance comparison, and finally analyze scalability across different network sizes.

The simulation considers a CF-mMIMO network deployed over a 500×500 m^2 area. Table I summarizes the key parameters. APs and CPUs are distributed uniformly at random, while users follow a clustered distribution to model hotspot scenarios. The wireless channel follows a three-slope path loss model with log-normal shadow fading. Each user is served by $D = 4$ APs selected based on large-scale fading coefficients.

The GNN encoder uses two graph attention layers with 4 attention heads, producing 32-dimensional embeddings. Each agent's policy network has two hidden layers with 64 neurons. Training is performed over 500 episodes using PPO with generalized advantage estimation. Results are averaged over 50 test episodes.

We compare the proposed GNN-CTDE against two baseline methods. *Closest CPU* assigns each AP to the geographically nearest CPU, minimizing backhaul path length but ignoring load distribution. *Load-Balanced Greedy* sequentially assigns each AP to the least loaded CPU, achieving near-optimal balance through explicit load tracking.

TABLE II
PERFORMANCE COMPARISON ($M = 50$, $K = 20$, $C = 4$)

Method	Sum Rate (bps/Hz)	Min Rate (bps/Hz)	Load Std
Closest CPU	274.2	10.72	3.32
Greedy	272.2	10.62	0.58
GNN-CTDE	**274.5**	**10.82**	**0.74**

Table II presents the main performance comparison at $M = 50$, $K = 20$. The Closest CPU method achieves sum rate of 274.2 bps/Hz but exhibits poor load balancing with standard deviation of 3.32. This imbalance occurs because AP density varies spatially, causing some CPUs to handle significantly more APs than others. The Greedy method achieves excellent load balance (std = 0.58) but relies on hand-crafted rules.

GNN-CTDE reduces the load standard deviation to 0.74, representing a 78% improvement over Closest CPU while approaching the Greedy baseline. Importantly, GNN-CTDE achieves this through a learned policy without explicit load-tracking logic. The sum rate of 274.5 bps/Hz is comparable to Closest CPU, confirming that load balancing does not compromise throughput.

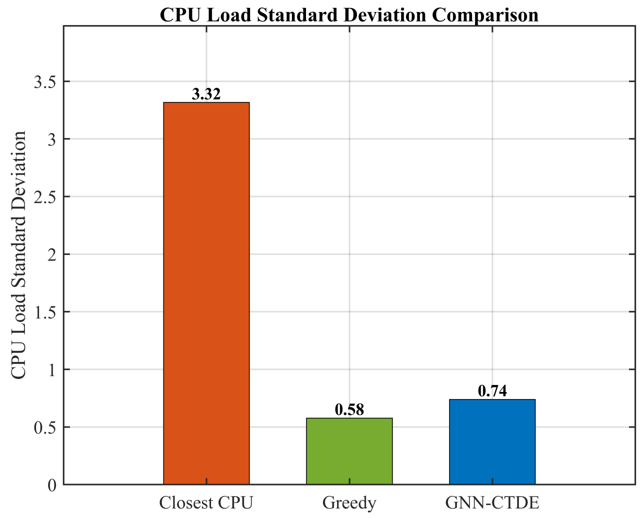


Fig. 3. CPU load standard deviation comparison

Fig. 3. visualizes the load balancing performance. The substantial gap between Closest CPU (3.32) and the other methods highlights the importance of load-aware assignment. GNN-CTDE achieves 78% of the gap closure between Closest CPU and Greedy.

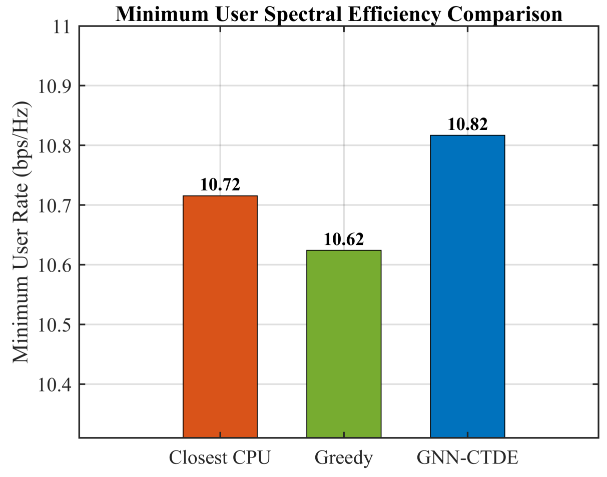


Fig. 4. Minimum user spectral efficiency. Higher values indicate better fairness for worst-case users.

Fig. 4. shows the minimum user rate, which reflects fairness among users. GNN-CTDE achieves 10.82 bps/Hz, outperforming both Closest CPU (10.72) and Greedy (10.62). This improvement results from the composite reward function that balances throughput, load distribution, and user fairness.

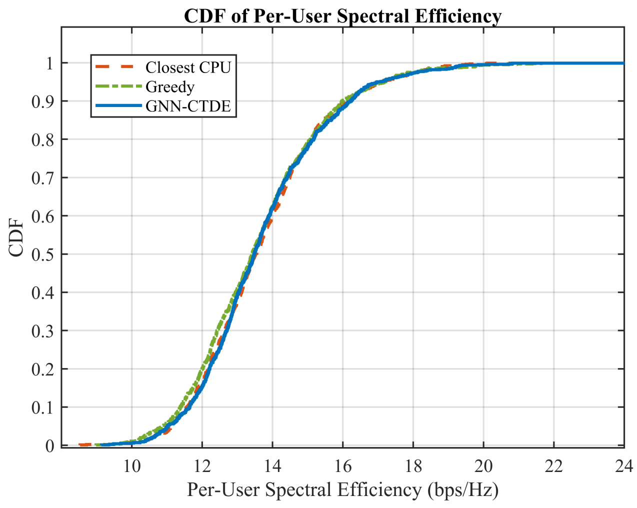


Fig. 5. CDF of per-user spectral efficiency.

Fig. 5. presents the cumulative distribution function of per-user spectral efficiency. All three methods exhibit similar distributions in the mid-to-high rate region, confirming that GNN-CTDE maintains overall throughput. The slight rightward shift at lower rates indicates improved service for disadvantaged users.

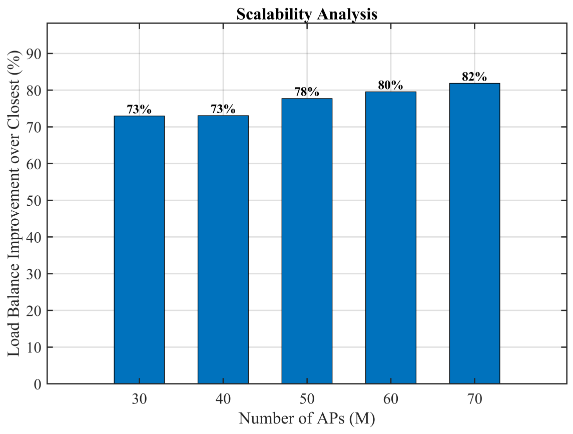


Fig. 6. Load balance improvement over Closest CPU across different network sizes.

To evaluate scalability, we trained and tested GNN-CTDE across network sizes from $M = 30$ to $M = 70$, scaling users as $K = 0.4M$. Fig. 6. shows the load balancing improvement over Closest CPU. The improvement increases from 73% at $M = 30$ to 82% at $M = 70$, demonstrating that the learned policy generalizes effectively to larger networks. This scalability stems from the graph attention mechanism, which captures local interference patterns regardless of total network size.

V. Conclusion

This paper presented a GNN-based multi-agent reinforcement learning framework for AP-CPU assignment in cell-free massive MIMO networks. The proposed GNN-CTDE approach employs graph attention layers to encode network topology and a sequential action selection mechanism for implicit coordination among agents. Simulation results demonstrated 78% improvement in load balancing at $M = 50$ compared to distance-based assignment, with the learned policy achieving the highest minimum user rate among all methods. Scalability experiments confirmed consistent improvements of 73–82% across networks ranging from 30 to 70 APs. Future work will investigate dynamic scenarios with user mobility and joint optimization with power control.

VI. Acknowledgment

This research was supported by the Regional Innovation System & Education (RISE) Glocal 30 program through the Daegu RISE Center, funded by the Ministry of Education(MOE) and the Daegu, Republic of Korea.(2025-RISE-03-001). Additionally, this research has been supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT), (NRF-2022R1A2C1003620).

References

[1] H. Q. Ngo, G. Interdonato, E. G. Larsson, G. Caire, and J. G. Andrews, “Ultradense cell-free massive mimo for 6g: Technical overview and open questions,” *Proceedings of the IEEE*, vol. 112, no. 7, pp. 805–831, 2024.

[2] M. Ajmal, M. A. Tariq, M. M. Saad, S. Kim, and D. Kim, “Scalable cell-free massive mimo networks using resource-optimized backhaul and pso-driven fronthaul clustering,” *IEEE Transactions on Vehicular Technology*, 2024.

[3] S. Buzzi and C. D’Andrea, “Cell-free massive mimo: User-centric approach,” *IEEE Wireless Communications Letters*, vol. 6, no. 6, pp. 706–709, 2017.

[4] M. Ajmal, A. Siddiqa, M. A. Tariq, M. M. Saad, and D. Kim, “Dynamic backhaul clustering for enhanced scalability in cell-free massive mimo networks,” in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, pp. 1735–1741, 2024.

[5] Z. Li, F. Göttsch, S. Li, M. Chen, and G. Caire, “Joint fronthaul load balancing and computation resource allocation in cell-free user-centric massive mimo networks,” *IEEE Transactions on Wireless Communications*, vol. 23, no. 10, pp. 14125–14139, 2024.

[6] Y. Tsukamoto, A. Ikami, T. Murakami, A. Amrallah, H. Shinbo, and Y. Amano, “Scalable ap clustering with deep reinforcement learning for cell-free massive mimo,” *IEEE Open Journal of the Communications Society*, 2025.

[7] D. Pereira-Ruisánchez, M. Joham, Ó. Fresnedo, D. Pérez-Adán, L. Castedo, and W. Utschick, “A gnn-based approach to ap cooperation cluster formation in cell-free massive mimo,” in *2025 IEEE 101st Vehicular Technology Conference (VTC2025-Spring)*, pp. 01–07, IEEE, 2025.

[8] L. Wang, C. Chen, J. Zhang, and C. Fischione, “Learning-based joint antenna selection and precoding design for cell-free mimo networks,” *IEEE Transactions on Vehicular Technology*, 2025.

[9] E. Björnson and L. Sanguinetti, “Scalable cell-free massive mimo systems,” *IEEE Transactions on Communications*, vol. 68, no. 7, pp. 4247–4261, 2020.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.